Performance Metrics for Embedded Parallel Pipelines

M. Fleury, A. C. Downton and A. F. Clark

Vision & Speech Architectures Group

Department of Electronic Systems Engineering, University of Essex,

Wivenhoe Park, Colchester, CO4 3SQ, U.K

tel: +44 - 1206 - 872817

fax: +44 - 1206 - 872900

e-mail fleum@essex.ac.uk

Abstract

A statistical approach to performance prediction is applied to a system development methodology for pipelines comprised of independent parallel stages. The methodology is aimed at distributed memory machines employing medium-grained parallelization. The target applications are continuous-flow embedded systems. The use of order statistics on this type of system is compared to previous practical usage which appears largely confined to traditional Non-Uniform Memory Access (NUMA) machines for loop parallelization. A range of suitable performance metrics which give upper bounds or estimates for task durations are discussed. The metrics have a practical rôle when included in prediction equations in checking fidelity to an application performance specification. An empirical study applies the mathematical findings to the performance of a multicomputer for a synchronous pipeline stage. The results of a simulation are given for larger numbers of processors. In a further simulation, the results are extended to take account of waiting-time distributions while data are buffered between stages of an asynchronous pipeline. Order statistics are also employed to estimate the degradation due to an output ordering constraint. Practical illustrations in the image communication and vision application domains are included.

Index terms: performance prediction, parallel pipelines, real-time systems, order statistics

1 Introduction

Continuous-flow embedded systems are an important class of engineering system for which parallel solutions have much to offer. There is often a latency and/or output ordering constraint imposed upon the output in addition to the usual throughput constraint. Examples of such parallel systems can be found in vision [1], radar [2], speech processing [3], and data compression [4]. In order to meet 'soft' real-time guarantees these example systems, conceived independently of each other, all share a common architecture.

The chief building block of that architecture is the data farm. A processor or data farm [5] is a programming paradigm involving message-passing in which a single task is repeatedly executed in parallel on a collection of initial data. Data-farming is a frequent paradigm in parallel processing [6] and appears in numerous guises: some (network-of-workstations) NOW-based [7]; some based on dedicated multicomputers [8]; some constructed as algorithmic skeletons [9]; and some implicitly invoked in parallel extensions of C++ [10].

In the case of continuous-flow systems, a pipeline has the ability to hide serial bottlenecks within a stage if the application can be partitioned in a reasonably balanced manner, thus increasing the exploitable parallelism. The idea of combining stages, each with internal parallelism, already occurs in multi-pipelines subsequently applied to systolic designs [11]. The 'Enterprise' system [7], and [12] are programming environments in which farms, if need be, can be combined in a pipeline on general-purpose processors. In the field of vision and image analysis, pipeline design methodologies have been proposed [13] with internal parallelism [14].

Motion estimation, with a nine-stage continuous-flow parallel pipeline, is the example application considered in [15], where the main concern is the logical partitioning of processors across the pipeline to achieve optimal throughput on an Intel iPSC/2. Airborne real-time STAP (space-time adaptive processing) radar is an example application consisting of five independent stages: Doppler filter processing, weight computation, beamforming, pulse compression, and CFAR (constant-false-alarm-rate) processing which is being implemented as a parallel pipeline on the Intel Paragon [16]. In both these applications, a further stage of pro-

cessing is likely to occur in which the results of the early data-reducing stages are analysed. Certainly, this was the case for the earlier target-tracking radar system reported in [2].

Given the ubiquity of farms, pipelines, and multi-pipelines, a system development methodology which would assist application engineers 'in-the-field' to prepare designs involving parallelism for continuous-flow systems has been proposed. Pipelined Processor Farms (PPFs) [17] targets multi-algorithm applications with some irregular algorithmic sub-components. A PPF is a linear pipeline with a single backplane. The linear pipeline is common [15], and in Section 2, ways to transform some pipelines to a linear form are considered. Deterministic (constant algorithmic time complexity) applications are already well-served by a system development methodology for systolic arrays [18]. Regular algorithms with a limited amount of branching are suitable for PPF, and indeed have been tackled as sub-components of an application [19, 20], but when considering a complete application in vision, PPF's principal target domain, it is usually the case that there will be at least one irregular component when the system is viewed as a whole. When a whole system is analysed the composition of algorithms is of concern and not simply individual algorithmic behaviour. However, irregular algorithms which involve global access to a common data structure are not suitable for PPF as the essence of the data farm paradigm is strictly local computation. For example, compare the parallel pipeline for speech processing reported in [3] in which the elements of a decoding network have been decomposed and are suitable for PPF development to the integrated speech decoder network reported in [21], which was unsuitable for the PPF approach.

PPF offers a constrained design model, deliberately avoiding a proliferation of solutions which can be confusing to the new-comer to parallelism. In PPF, there is a single primary pipeline which is capable of incorporating temporal, data, or algorithmic parallelism within individual farms.[1] The need for such diversity arises for systems composed of a number of independent algorithms. This study is concerned both with temporal farming in which a set of indivisible work jobs are distributed to a set of worker processors, and data farming in which

[1]Data parallelism in this instance is performed at run time and is medium-grained as opposed to the data parallel programming model which commonly uses fine-grained parallelism, set-up at compile time.

a job is decomposed into tasks for scheduling convenience. Temporal farming is defined by us as being where no parallel decomposition of the algorithm takes place, for example if a set of video frames were processed in parallel using the originally specified sequential algorithm. Data parallelism would be where sub-frames were processed in parallel, requiring alteration of the sequential algorithm. Work farming is a generic term describing both temporal- and data-farming.

For the purposes of this paper, a job is defined as a finite set of tasks. For a continuous-flow system a job arises if for the purpose of measurement the flow was halted at some time. In the class of applications of interest, tasks can frequently be combined into chunks (nomenclature adopted from the performance literature cited). For example, we have parallelized handwritten postcode recognition [1], with characters (tasks) within a postcode (UK zipcode) (chunk) from a file of postcode images (job); a hybrid video encoder [22], with macro-blocks (tasks) of a video row (chunk) taken from a sequence of frames (job); and 2-D frequency transforms [23], with rows (tasks) in a sub-image (chunk) from a batch of images (job).

PPF has been applied to low-cost distributed-memory machines using a message-passing programming paradigm. Understanding the performance of first generation multicomputers has involved accounting for a store-and-forward method of communication, possibly by means of linear programming [24]. In a detailed study [8], data farming was applied with constant task durations, as is common in numerical analysis algorithms. An extension employing mean-values was developed. In [8], unequal paths through the network topology were the principle cause of perturbation, though congestion, which is non-linear, and the system of message-passing were also an issue. Store-and-forward communication was employed as has been common on first-generation multicomputers. With some refinements, the linear programming method was adopted. Linear-programming gives a lower-bound to performance based on the raw physical bandwidth of the processor interconnect and message set-up time at the processor. For the regular algorithms investigated and for single farms (and single algorithms) there is some prospect of reaching ideal performance at the lower bound.

However, PPF has outlived the era of first-generation multicomputers. What are the char-

acteristics of the PPF communication model for second-generation multicomputers? Firstly, a number of high bandwidth interconnects are becoming available, implying that even for bandwidth-hungry applications such as image processing the main issue will not be reaching the physical limits of the interconnect. The Intel Paragon (message-passing) network has a full-duplex bandwidth of 175 MBps (M bytes per sec.). Myricom's Myrinet network [25] has a peak duplex bandwidth of 160 MBps and is eminently suitable for large-scale embedded applications. Secondly, with the advent of generalized communication methods such as the routing switch [26], the 'fat' tree [27], and 'wormhole' routing [28] communication variance is of diminishing importance. Wormhole communication has reduced the message set-up component. Routing congestion can be alleviated either by a randomized routing step [29], or by a changing pattern of communication generated through a latin square [30].

The data-farm implies all communication is between farmer and worker processors and not outside the farm. First-generation embedded applications always had the option of a fast bus between farmer processors [2] to ease a bottleneck between farmers on the pipeline. It is also possible to place the farm workers between two farmers, when modularity is less of an issue, for example in [31]. The second farmer then collects the results directly from the first farm's workers, before collating results and passing the results to its own workers. However, a form of statistical multiplexing of work, not demand-based data-farming would seem more appropriate for these data-flow designs. Statistical multiplexing is outside the scope of this paper.

Considering the changes since PPF's inception, it appears that a single metric, the mean communication latency, may suitably characterize PPF communication performance. For predicting the overall time of a job involving a large number of task-bearing messages, the mean latency is sufficient provided the distribution of transfer times is infinitely divisible, as is the case for deterministic, normal, or exponential message latency distributions. Obviously a constraint given by the physical bandwidth still exists but we consider the case if that limit is not approached. The advantage for the system developer is that the behaviour of the algorithm becomes central, with communication characteristics remaining stable and

decoupled from the algorithm. It may appear that with this communication model there is no problem to consider but in fact in [32] with a similar communication model the maximum efficiency asymptotically approaches a value of 0.4. The non-asymptotic version of the model is discussed in Section 5.

We set out to derive the performance PPF systems could expect on second-generation multicomputers. The aim was to find an analytical performance model. On the available eight module distributed-memory research machine, the Paramid [33], in which wormhole communication is simulated by a virtual channel system [34], the measured communication time was found to be a linear function of message size. Packetization and message aggregation for a common destination are used to reduce message transit variance. Each Paramid module has a coprocessor dedicated to message-passing but no hardware support for shared memory. The Paramid's interconnect bandwidth is limited to 20 Mbps links necessitating simulation of high-bandwidth interconnects by sending small messages, and including a notional communication latency. In Section 6, experiments with the Paramid are reported.

Traditional non-uniform memory access (NUMA) multiprocessor machines also have been modelled [35] using mean latency as the sole communication metric. We assume a traditional NUMA architecture to be one in which a multistage network, such as an Omega, Banyan, or Benes network, lies between the processors and the memory modules, examples of which are the Denelecor HEP, the NYU Ultracomputer, the IBM RP3, and the BBN Butterfly [36].[2] On traditional NUMA machines a task queue is commonly kept [38]. One machine will perform sequential phases of an algorithm while independent loop iterations (identified by a parallelizing compiler) are scheduled between available processes. Though our interest stems from traditional NUMA machines it should be noted that parallelising compilers preserve the master/slaves arrangement on non-traditional NUMA machines [39].

[2]The Cray T3E is a non-traditional NUMA design with a 3-D grid network between processors but still a shared address space. The SGI Origin and Sequent NUMA-Q are non-traditional cache-coherent variants of NUMA, *i.e.* ccNUMA with additional hardware-assist for cache coherence, again with a shared address space.[37]

On tightly-coupled machines, even when deterministic numerical analysis algorithms are run, it has been found that system perturbation delays synchronization between stages of the algorithm [40]. Similarly on NUMA machines, algorithmic and/or system perturbation is found. Rather than employing static scheduling at compile time, 'self-scheduling' is a means of smoothing out the perturbations. An advantage of self-scheduling is that the operating system may be by-passed provided the problem of 'hot-spot' access to the iteration counter is solved [41]. In [35], the claims of various statistical models for the performance of 'self-scheduling' of loop iterations have been tested out on a BBN Butterfly TC2000. The models tested turned out to be broadly accurate.

On traditional NUMA machines it is also possible to partition the available address space. Where there is locality of data reference, message-passing between the partitions under the Chrysalis operating system has been shown to compete favourably on the BBN Butterfly GP-1000 [42]. Stemming from the RP3, there has been a convergence in physical design between NUMA machines and distributed-memory machines, for example between the Cray T3x family and the Intel Paragon which has enabled message passing to be used on machines. For example, For example, 'Active Messages', light-weight user messages, have also successfully been applied [43] to ccNUMA multiprocessors.

Therefore, it occurs to us that performance models for traditional NUMA machines may equally apply to the distributed memory systems used by PPF. These performance models refer to traditional NUMA machines employed without message-passing, whereas PPF systems have been deployed on machines dedicated to message-passing. The purpose of this paper is to report tests in support of that initial insight.

Self-scheduling is similar in concept to demand-based farming, which is employed in PPF to schedule tasks in data-parallel mode. In demand-based farming, where the number of tasks in a job is much greater than the number of processes, messages returning processed work (or a suitable token) form implicit requests for more work from a central 'farmer'. Neglecting congestion, the computation time of an individual task ideally should at least exceed the maximum two-way data transfer time from worker process to farmer process. Local buffers

allow a worker process to work from a buffer while a request for further work is satisfied by the farmer, thereby overlapping computation with communication. As a worker must have some work before it can request more, initially, all processes are loaded with some work from the central farmer process, *i.e.* a static scheduling phase. The purpose of the initial phase is to ensure all local buffers are full. Each processor will normally host a single process, except in the case of the farmer process which may share its processor with a worker process.

Our application of a NUMA performance model to PPF systems assumes the following points:

- There is no global operating system on distributed-memory machines as there is on traditional-NUMA machines with a global address space [44]. However, a central farmer process for each farm can take the place of the operating system in respect to scheduling.

- PPF applications are medium-grained, without the comparatively fine-granularity of NUMA loop iterations. This is because the access latency to remote memory is an order of magnitude greater on distributed-memory machines than it is on shared-memory machines [45]. However, the difference in granularity should merely produce a scaling effect on the results. In regard to performance modelling, data-farming systems might properly be considered a sub-class of NUMA though in point-of-fact the performance literature, *e.g.* [46], has treated data-farms as a separate class.

- The traditional-NUMA performance models were tested on numerical analysis algorithms and not on vision applications. (Though recent experiments employ a variety of benchmarks [47]). Vision applications may involve irregular algorithms, without significant loops. Image-processing applications tackled in PPF may involve high bandwidths. However, communication latency can often be masked by local buffering of data, which will work even when latency masking through 'parallel slackness' [29] is unavailable as an option. In general, the number of processors in vision embedded systems on any one farm is unlikely to be large. In fact, each stage commonly consists of a few processors (about four). Where PPF applications are parallelized from sequential versions, the

task duration distributions may be available as well as second-order statistics.

Provided account is taken of the differences between systems, the same performance prediction and scheduling schemes might be used for both. Preliminary results in support of this conclusion were reported in [48]. This paper also has a concern with finding the maximum per task pipeline traversal latency across PPF systems. Previous work in this field [14], being concerned with balancing pipelines, relied on queueing theory, and found mean-valued system behaviour.

In Section 2, the form of PPF systems is identified. In Section 3 a number of suitable metrics from the statistics of extremes are assessed. Section 4 discusses the performance indicators needed for PPF and includes application examples. Section 5 is an analysis of run-time prediction equations, which are extensions of the basic metrics. Section 6 is the result of an empirical study on the utility of the metrics. Section 7 reports the results of a simulation to verify that the metrics scale-up. Section 8 is an extension of the performance model which utilises queueing theory to find the inter-stage maximum waiting time for asynchronous pipelines. The same statistical approach can also be employed to predict performance degradation from output ordering constraints, Section 9, which also includes an application example. Finally, Section 10 is a summary with some concluding remarks.

2 PPF System Types

PPF systems can be characterized as either asynchronous or synchronous pipelines. If there is just one pipeline stage that has a synchronization point then that stage will affect the work flow on all subsequent stages. In PPF, a synchronization point may arise from either an algorithmic constraint, or a feedback path. An algorithmic constraint is typically the completion of all tasks within a single job instance. An ordering constraint is a form of algorithmic constraint in which a subset of tasks within a job must complete before that subset of tasks may proceed to subsequent processing. A feedback path can be modelled as a pipeline stage governed by an algorithmic constraint though no such physical stage occurs. A

folded-back pipeline stage is a way of avoiding a synchronization constraint by using a stage for two different processing activities. Unfortunately, in practice the time characteristics of the two stages seldom coincide. Again the delay can be modelled by an additional notional stage. In Figure 1(A) two feedback loops are shown which in Figure 1(B) have been replaced for modelling purposes by nominal stages representing delay. In Figure 1(C) a folded-back pipeline, where algorithms 1 and 3 are performed at the same stage, is replaced in Figure 1(D) by a feedback loop with the folding removed. Then in Figure 1(E), the remaining feedback loop is removed so that the pipeline can be modelled as a linear pipeline. Figure 1(B) & (E) show two pipelines in normal form. Once in normal form, a pipeline is amenable to systematic performance analysis. Across a series of consecutive pipeline stages, traversal latency is additive. Therefore, a prime objective is to be able to find the traversal latency at any single stage.

Asynchronous pipelines are those in which no synchronous constraints apply. Asynchronous pipeline segments may exist as pipeline segments within a synchronous pipeline. Figure 2 shows three examples of pipelines containing both synchronized and asynchronous segments. Figure 2(B) & (C) require normalization by removal of feedback lines. A synchronized segment may contain any finite number of asynchronous segments, Figure 2(B) & (C). The inclusion of an asynchronous segment within a larger synchronous segment is explained by the effect of a later synchronous constraint on the output from a portion of a pipeline that has no constraints. The global traversal latency of an asynchronous segment must be found as the overall latency cannot be built up in a stage-by-stage fashion. Once the maximum traversal latency of an asynchronous segment is found, it can be treated as an estimate for a single stage within a synchronous pipeline.

The H.261 low bit-rate hybrid video encoder [49] has been parallelized by the PPF methodology [22] to enable further rapid algorithmic prototyping. Figure 3 shows a block structure that can be reduced to a linear pipeline by suitable combination of functions, Figure 4. The linear pipeline is largely synchronous due to the remaining feedback loops. A static timing analysis, Table 3, gave the ratio of times spent within each of the top-twenty component

functions within a sequential implementation. Though two sets of functions were candidates for data-farming, with video macro blocks as the tasks, these sets of functions could not be processed in parallel within a pipeline. Therefore, the pipeline was folded back with the middle of the three stages in Figure 5 alternating processing between two alternative sets of functions. In this pipeline, the final stage also alternates its activity. The pipeline is approximately balanced by means of data-farming within the middle stage, though due to the disparity between the times for the two alternative sets of functions there will be processor idling. Conversely, the partitioned pipeline allows the sequential components at stages one and three to be processed in parallel with the other stages.

The handwritten postcode (UK zipcodes) recognition PPF example [1] is a three-stage pipeline comprising preprocessing of the digitised postcode image, classification on a per-character basis, and address dictionary search to find a best match ($\approx 80\%$ accuracy for a single candidate postcode). The first two asynchronous stages employ data parallelism, with character-based processing as the task. However, the need to perform a dictionary search with a complete postcode, *i.e.* temporal parallelism, imposes a partial ordering constraint. All characters must be re-assembled into a postcode before performing the dictionary search for the identified postcode. In fact, there was a further constraint because the postcodes must leave the pipeline in the same order that they entered, though buffering is possible. If the dictionary stage is also modelled as asynchronous then the expected degradation in performance through postcode ordering can be modelled by the technique in Section 9. In Figure 6, the handwritten postcode recognition PPF is shown with just one worker process at the dictionary stage to achieve optimal performance. However, this arrangement is an artifact caused by the limited processor modules on the test machine, and in Section 9 scaling experiments for the dictionary stage are reported.

3 Performance Metrics

A common theme to performance prediction for 'self-scheduling' is the use of order statistics, when one is usually interested in the extremal orders. Order statistics reflect the individual distributions of p random variables selected from a population and placed in ascending order. Order statistics have been employed in directed acyclic graph (DAG) models of parallelism, stemming from [50].[3] The general properties of series/parallel graphs, SPG, of the DAG variety with unconstrained numbers of nodes and probabilistic branching have been studied from the standpoint of queueing theory in [51]. [15] is a practically-oriented study of the SPG model for parallel pipelines, though not using queueing theory or order statistics. Queueing theory is not normally helpful for the performance of individual applications as it gives rise to means not maxima. The linear form of pipelines in PPF means that the wider generality of the SPG model is not helpful. In PPF, a tight upper-bound is sought to check that real-time constraints are met. However, the mean of the maximum or other common averages such as the mode and median, are not necessarily the correct statistic when dealing with extremal statistics. For example, the characteristic maximum, considered in Section 3, may be a more suitable statistic.

The maximum duration of any task, viewed stochastically, can be found from extremal statistics, which are concerned with probability distribution tails.[4] A number of distribution-free estimates for the behaviour of distribution tails are available [52]. The underlying notion is that distribution of the tails may be grouped into common families of distribution. Distribution-specific estimates are also possible in PPF though this involves extra statistical pre-processing to establish any distribution, Section 4. It is also not always the case that 'real-life' distributions can be confidently matched to any one classical distribution, though broad classifications of symmetry or asymmetry are of value. However, exact results are also

[3] Earlier work [11] was also concerned with asynchronous algorithms on small-scale multiprocessors.

[4] A distribution tail is a common informal term applied to unimodal distributions, *e.g.* [32], meaning: that portion of the area under the curve of a probability density function beyond some point at a significant distance from the mode.

useful for checking the accuracy of estimators.

3.1 Order Statistics

In this section, the fundamental results of extremal statistics estimators are established. Certain equations are designated a name for easy reference in Sections 6 & 7.

Consider a set of p continuous independent and identically distributed (i.i.d.) random variables (r.v)., $\{X_{i:p}\}$, with common probability density function (pdf) $f(x)$, and constrained such that $-\infty < X_1 < X_2 < \ldots < X_p < \infty$.[5] Consider further

$$F_{i:p}(x) = \sum_{m=i}^{p} \binom{p}{m} (F(x))^m (1 - F(x))^{p-m}, \tag{1}$$

which is equivalent to P(i or more variates $\leq x$), with $P(\cdot)$ returning the probability. $F_{i:p}(x)$ is the cumulative distribution function (cdf) of the ith order statistic out of p with variate x. From (1) or otherwise, the pdf of the maximum order statistic [53] is derived as

$$f(x_{p:p}) = pF^{p-1}(x)f(x). \tag{2}$$

Therefore, the mean is

$$E[X_{p:p}] = \mu_p = p \int_{-\infty}^{\infty} xF^{p-1}(x)f(x)dx, \tag{3}$$

where E is the mathematical expectation operator, and similarly the mean of the minimum is

$$E[X_{1:p}] = p \int_{-\infty}^{\infty} x(1 - F(x))^{p-1} f(x)dx.$$

The range is the difference between the two extremes. Consequently, the mean of the range of p random variables, w_p, is

$$E[w_p] = \mu_{w(p)} = p \int_{-\infty}^{\infty} x \left(F^{p-1}(x) - (1 - F(x)\right)^{p-1}) f(x)dx. \tag{4}$$

[5] We use the standard convention that X is the r.v. and x its value.

Provided that the mean, μ, and the standard deviation (s.d.), σ, of the common cdf, $F(x)$ exist, it is possible to use the calculus of variations[6] to find the maximum of μ_p [55] (with region of support for the pdf from 0 to 1):

$$\max(\mu_p) = \mu + \sigma\frac{p-1}{\sqrt{2p-1}}. \tag{5}$$

The variational constants are not resolved if the calculus (with the same region of support) is used to find the maximum of $\mu_{w(p)}$ but on converting to a standardized variate, and with $\sigma = 1,\ \mu = 0$, the result is

$$\max(\mu_{w(p)}) = p\sqrt{\frac{2(1-\epsilon_p)}{2p-1}},$$

where $\epsilon_p = (p-1)!^2/(2p-2)!$, which vanishes as 2^{-2p}. For a symmetrical distribution the mean of the largest value, which is $\max(\mu_{w(p)})/2$, is $1/\sqrt{2}$ down on the equivalent value given by (5). However, the mean grows more slowly, as $\sqrt{p}/2$ and not $\sqrt{p/2}$.

The standardized upper bounds that arise for the mean of the maximum value are plotted in Figure 7. Suppose that p tasks are started at the same time, then the mean of the finishing time of all the tasks is either asymptotically bounded by μ_p or $0.5\mu_{w(p)}$ depending on the type of the task duration cdf. However, it is not difficult to convert from asymmetrical to symmetrical distribution since by grouping sufficient tasks into chunks the sum will approach a normal distribution. This observation follows by the well-known Central Limit theorem, *i.e.* distribution of $p^{-\frac{1}{2}}(S_p - n\mu)$ approaches a normal distribution, for S_p the sum of a sequence of p r.v. with finite variance and arbitrary distribution.

In fact, exact results are available by solving (3). Notably, for the exponential cdf $(1 - e^{-\lambda x},\ x > 0$, here $\lambda = 1)$,

$$\mu_p = \sum_{i=2}^{p}\frac{1}{i} = \ln p + \gamma + O(p^{-1}), \tag{6}$$

with γ being Euler's constant $(0.5772157\ldots)$[7], and where as usual $O(f(n))$ is the set of

[6]There is an alternative derivation using the Cauchy-Schwartz inequality [54].

[7]Equation 6 will be recognized as a variant of Riemman's zeta function.

functions of growth rate order $f(n)$for some variate n. Additionally, for the standard normal distribution [56, pp. 374-378]

$$\mu_p = (2\ln p)^{1/2} - \frac{1}{2}\frac{1}{(2\ln p)^{\frac{1}{2}}}(\ln(\ln p) + \ln 4\pi - 2\gamma) + O\left(\frac{1}{\ln p}\right),$$

designated 'max', and for the Uniform distribution on $[0, 1]$

$$\mu_p = \frac{p}{p+1}.$$

The distribution-specific standardized means are plotted in Figure 7 so that the relationship to the maximized means is evident. The maximised means clearly represent upper bounds. In general, the distribution may be unknown or μ_p may be difficult to derive and some distributions may approach the upper bounds slowly. Naturally, as p increases the possibility increases of a large timing pushing the mean upwards away from the majority of the timings.

3.2 An Asymptotic Distribution

Suppose

$$G(x) = F^p(x)$$

is the probability that, out of p observations, all are less than x. Then the asymptotic distribution has the 'stability' property that

$$G^p(x) = G(a_p x + b_p), \tag{7}$$

since the form of original distribution is not altered by applying a linear transformation. If a_p is set to one, after some work, it is found that

$$G_{(1)}(x) = \exp(-e^{-x}), \tag{8}$$

which is the first asymptotic distribution of G. It can be shown [52] that the normal and exponential distributions have asymptotic distributions of this type for which all moments

exist (which is a necessary but not sufficient condition). By integrating (3) for $G_{(1)}(x)$ in (8) and converting to standard form, it is found that

$$E[G_{(1)p:p}] = \mu_G = \ln p(\sqrt{6}/\pi),$$

which will be designated 'asymptotic'. It would appear that μ_G represents a suitable estimate for the maximum value. Unfortunately, the standard normal distribution asymptotic behaviour at large values of p converges slowly to that of the double exponential distribution (Figure 8).

By setting $b_n = 0$ in (7) the second and third asymptotic distributions are found as

$$\lim_{p\to\infty} G_{(a)}(x) = \exp\left(-x^{(-1)^{a-1}k}\right),\ a = 2, 3, \tag{9}$$

for some constant $k > 0$. The second asymptotic distribution is a fit for a cdf with no or only a few moments such as respectively the Cauchy distribution or the polynomial distribution, common in modelling bursty traffic but not often found in computing applications. The asymptotics of $F^p(x)$ where $F(x) = x$, *i.e.* the uniform distribution, is an example of $G_{(3)}$ for which the distribution is bounded in some way. Note that some distributions fall into none of the three asymptotic categories.

3.3 The Characteristic Maximum

Because μ_p may be difficult to find and because both μ_p and μ_G may present too loose a bound another measure, the characteristic maximum, m_p, may act as an estimate. However, the characteristic maximum is not an upper bound but is most closely associated with the mode or most popular value for the maximum. Previous work in applying these results to parallel computation has not emphasized this relationship. The similarity between the mode of the maximum and m_p is very noticeable in the case of the normal distribution.

Define m_p such that for a cdf H

$$p(1 - H(m_p)) = 1, \tag{10}$$

i.e. out of p timings just one is greater than m_p. Consider,

$$\lim_{x\to\infty}\frac{h(x)}{1-H(x)} = -\lim_{x\to\infty}\frac{h'(x)}{h(x)},$$

where the last equality arises from L'Hôpital's rule, provided $h(x)$ and $1-H(x)$ are small for large x. By equating the derivative of the pdf of (2) to zero the mode of the distribution of the maximum is found to satisfy

$$\frac{(p-1)h(x_{mode})}{H(x_{mode})} = \frac{-h'(x_{mode})}{h(x_{mode})}$$

by substituting (3.3) into (3.3) it will be seen that

$$H(x_{mode}) \approx H(m_p) = 1 - 1/p.$$

If H is the distribution of the sum of a number of r.v. of a distribution with finite second moment, an estimate of m_p for a normal distribution [57] might be used:

$$\sqrt{2\ln p - \ln(\ln p) - 3} < m_p < \sqrt{2\ln p - \ln(\ln p)}, p \geq 5. \tag{11}$$

Inequality (11) arises because for some cdf

$$E[X_{p:p}] = m_p + n\int_{m_p}^{\infty}(x-m_p)f(x)dx$$

and in particular for a standard normal pdf, $\phi(\cdot)$, the R.H.S. of (3.3) is $E[X_{p:p}] = n\phi(m_p)$. A number of potentially useful approximations for the normal distribution are also demonstrated in [58, pp. 136–138]. Since for large values of x the normal distribution asymptotically approaches

$$H(x) = 1 - h(x)(1 - x^{-2} + 3x^{-4} - 15x^{-6} + \ldots)/x,$$

using (3.3)

$$\begin{aligned} p &\approx \frac{x_{mode}}{h(x_{mode})} + x_{mode}^{-2} \\ &\approx \sqrt{2\pi}x_{mode}e^{\frac{x_{mode}^2}{2}} + x_{mode}^{-2}. \end{aligned}$$

Solving for x,

$$x_{mode} \approx \sqrt{2\ln(p/(2\pi))}, x_{mode}^2/2 \gg \ln(x_{mode}), \tag{12}$$

which should be compared to (11).

The characteristic maximum of the exponential distribution is easily derived from (10)

$$m_p = \sigma \ln p, \tag{13}$$

(designated 'm_p') which should be compared to (6), where $\sigma = 1$. In fact, $m_p = x_{mode} < x_{med} < E[H_{p:p}]$ where x_{med} is the median of $H(x)$ an exponential distribution. The value of $E[H_{p:p}]$ is already given in standardized form in (6) and is almost within γ of m_p.

In [59], it is further proven that for large p, m_p approaches $E[H_{p:p}]$ from below provided H is a cdf with finite moments, is the cdf of a positive r.v., and has an increasing failure rate (IFR) (*cf.* (3.3)). An IFR distribution, H, is defined as:

$$\frac{h(x)}{1 - H(x)} \quad \text{monotonically increases with } x, x \geq 0. \tag{14}$$

IFR distributions, which are further referred to in Sections 5, are an alternative categorization to the three asymptotic categories.[8]

The variance of the first asymptotic distribution, (8), is given by

$$\sigma^2_{(1)} = (\pi/\sqrt{6}\alpha_p)^2,$$

where α_p is the value of the intensity function at $x = m_p$ (*i.e.* $ph(m_p)$). Therefore, since the intensity function of the double exponential distribution is an increasing function of p, the estimate improves with p. However, the variance of the second asymptotic distribution, (9), is given by

$$\sigma^2_{(2)} = m_p^2 \Gamma(1 - 2/k),$$

provided $k > 2$, k being a distribution-dependent parameter can be estimated from the coefficient of variation [55, p. 266] ($\Gamma(\cdot)$ is the Gamma function). Since for cdf bounded by (9) m_p increases as p, the value of (9) as an estimate is limited.

[8] Expression (14), a variant of the intensity function, is the probability function that given an event has occurred after x it will now occur. In general, the intensity function governs the convergence of a distribution's tail.

3.4 Sample Estimate

The foregoing estimates are based on population statistics. A simple bound on the sample statistics is easily derived:

$$\frac{(x_{p:p} - \overline{x})^2}{p-1} \leq s^2,$$

$\overline{x}$ and s being respectively the sample mean and s.d., is rearranged to yield

$$x_{p:p} \leq \overline{x} + s\sqrt{p-1},$$

which is designated 'sample'. The sample estimate is an upper bound to all previous estimates.

4 Performance Considerations for PPF

PPF systems are data-dominated systems which have soft real-time targets. Real-time performance is dependent on maximum latency and minimum throughput specifications. In order to meet a specification these should ideally be population statistics, evident from Section 3. Otherwise, a set of timings from representative test data can be made. Given a sequential version of an application, sections of code are timed in an isolated environment. The same sections of code are preserved intact as the kernel of the worker processing task in the parallel version. Counter-based profilers can give a timing that is independent of system load. A partition is provided between user code and system call code, which is useful when transferring between machines. However, the profiler available to us did not allow the global timing to be decomposed. Estimates of the time needed for small sections of code also can be made from source code but only in restricted circumstances due to the effects of compiler optimization [60]. Due to advances in compiler technology and as the i860 is a superscalar processor, we do not use this method. Instead, we timed code on a single processor within the parallel machine in order to cut out system load. Timings are assembled into a task duration histogram. The chi-square and Kolmogorov-Smirnov tests are well-known generalised methods to establish a fit to the histogram [61, pp. 39-52]. The use of such a method to fit a distribution is reported in [35].

Notice from Section 3, that for many task duration distributions, the maximum duration varies statistically with the number of processors as $O(c\sqrt{p})$, which may mean that increasing the mean throughput will increase the maximum latency, albeit at a slow rate. c is an arbitrary factor that will vary with the task scheduling system. If it were desired to find the value of c, this would be done empirically by taking a set of measurements for varying values of p and using nonlinear regression [62]. Demand-based scheduling can be optimised if tasks are grouped into uniform-sized chunks. One then minimizes an expression for the run time which includes the chunk size as a parameter. An alternative is to have a chunk size which decreases in time. However, for algorithms for which the data size increases with the problem size, buffering demands may make decreasing chunk sizes impractical.

5 Performance Prediction Equations

The raw estimates of the maximum task duration can be combined to form performance prediction equations. If there were a perfect parallel decomposition of the task durations, one expects the running time to be

$$d = \frac{n\mu}{p} + \frac{nh}{pk}, \tag{15}$$

where p is the number of processors in a farm, n is the number of tasks and k is the number of tasks in a chunk. h is a fixed per chunk overhead, which would include the cost of communication and any central overhead. As mentioned in Section 1, h can be safely assumed to be fixed if the overhead has an infinitely divisible distribution.[9] The first term is (numerator) the total run-time for finitely large n, acceptable as this is a continuous-flow system, divided by (numerator) the degree of parallelization with zero synchronization cost (*i.e.* 'perfect parallelization'). k is needed in (15) as, though there are n tasks, only n/k chunks are sent out.

[9] z has an infinitely divisible distribution if, for each positive integer n, there are i.i.d. random variables $\{z_1^n, z_2^n, \ldots, z_n^n\}$ such that $z = z_1^n + \cdots + z_n^n$ [63, p. 252]

In [64], a distribution-free upper bound was proposed based on (5). When the first processor becomes idle there are naturally $p-1$ active processors. The remaining time is

$$E[R_{ms}] = k\mu + \sigma\sqrt{\frac{k(p-2)^2}{2p-3}} + h,$$

i.e. as if the first processor finishes just when $p-1$ chunks are assigned but have not been transferred to the remaining processors. Therefore

$$E[T_{ms}] \leq d + E[R_{ms}],$$

which we designate 'M&S', after the names of the originators (M)adala-(S)inclair. It may seem odd that when combining jobs in a task that varying the number of jobs would make a difference to the time taken. Yet this is exactly what was implied by task scheduling experiments on an early multicomputer [65], because of the changing statistical properties when adding (convolving) distributions.

In [32], three main bounds occur based on finding m_p for different chunking regimes and predicated on IFR distributions. m_p is taken to be the time to finish after the last chunk has been taken. Notice that the s.d. of k tasks with common distribution is $\sigma\sqrt{k}$. An easily-derived upper bound for the time up to when the last chunk is taken is

$$E[T_{start}] \leq \frac{n-kp}{p}\left(\mu + \frac{h}{k}\right).$$

Now $d = E[T_{start}] + h$, where the extra h arises from sending the last chunk. Where $k \sim n/p$,

$$\begin{aligned} m_p &\approx k\mu + \sigma\sqrt{2k\ln p} \\ E[T_{KW_{large}}] &\approx d + \sigma\sqrt{2k\ln p} \end{aligned} \tag{16}$$

which we designate 'KWlarge', after the names of the originators, (K)ruskal-(W)eiss, and large because $k/\log p$ should be large. The result (16) should come as no surprise in view of (12). In fact, (16) is derived via the theory of large deviations [66, p. 184] which perhaps obscures the relevance of the result. However, (16) should be applied with care since it is not standardized yet it will be observed that there is no μ dependency in the remainder portion of (16).

A tighter bound can be found if $k \ll n/p$. This is the normal regime for demand-based farming. If $\sqrt{k}/p$ is small,

$$\begin{aligned} m_p &\approx k\mu + \sigma\sqrt{2k\ln\left(\frac{p\sigma}{\sqrt{k}\mu}\right)} \\ E[T_{KW_1}] &\approx d + \sigma\sqrt{2k\ln\left(\frac{p\sigma}{\sqrt{k}\mu}\right)}, \end{aligned} \tag{17}$$

which we designate 'KW1'. If $\sqrt{k}/p$ is large,

$$\begin{aligned} m_p &\sim k\mu + \sigma\frac{p\sigma^2}{\mu} \\ E[T_{KW_2}] &\approx d + \sigma\frac{p\sigma^2}{\mu}, \end{aligned} \tag{18}$$

which we call 'KW2'. This last bound would occur if, for large n, the chunk size k was much larger than p, hence the lack of k-dependency in the remainder term of (18).

The prediction equations of [32] are estimates which are only reliable for the appropriate regime. The bounds ideally require n, p and k to all be large. From the point of view of predicting the run-time, and not the optimal scheduling regime, k is a scaling factor. If one sets $k = \mu = \sigma = 1$ as a form of normalization it is found that equations 16 & 17 are the same. For an exponential distribution, which has the coefficient of variation (c.o.v) $\sigma/\mu = 1$, with $k = 1$ the two equations are also the same. This suggests that (17), since it is dependent on the c.o.v. as well as p and k, is the most reliable. Normalized versions of the prediction equations are plotted in Figure 9.

6 Results

6.1 Experiment Setup

The aim of our timing experiments was to check whether predictive equations that were successful for NUMA machines might be appropriate for our machine and ones similar to it. The r.v. considered in Section 3.1 provide a mapping from task durations to the real line, thus linking the prediction equations with the experiments. The random variables are i.i.d.

which is appropriate to data farming (Section 1) as there are no inter-task dependencies. The timing tests obviously do not verify large p behaviour, which is considered in Section 7.

The Paramid test machine has eight modules and was in single-user mode. Each module comprises an i860 computational engine with a transputer communication coprocessor. The farmer is placed on a transputer where it does not impede computation.[10] On a farm template a variety of statistical regimes can be parameterized, amounting to a parallel test-bed. Four hundred tasks were sent out on demand for each job, *i.e.* the equivalent of single task self-scheduling. Buffering was turned off and the message size was restricted to a 12 byte tag and 16 bytes data. The raw bandwidth of a transputer link is 20 Mbit/s. For larger task durations the i860 running at 50 MHz waited on a 1 μs clock. Where the wait resolution could not be improved, the task duration was given as a number of loop iterations. The task durations were timed during the parallel run so that the sample mean and s.d. could be found. Similarly, the sample mean start-up time and the sample mean message delivery time were formed. A software synchronized clock with worst-case error of 0.5 ms [67] was employed. The overall time was taken at the central task farmer. The prediction equations were then calculated using the mean start-up time as an extra term in the equations.

6.2 Prediction Results

Tests ranged over a number of distributions because it is thought that this approach to testing widens the generality of the tests. An alternative strategy would have been to simulate worst-case task distributions such as linearly increasing task size. In the tests, the job duration level varies across distributions according to choice of distribution parameters. However, the results are consistent for any one distribution.

The normal distribution is ubiquitous, for example it models the sum of tasks in a chunk. Both the small and large duration normal distribution test job durations were accurately predicted in all cases. Standard normal variates were transformed in a way that preserves the

[10] Notice that though the transputer is inherently multi-threaded, the i860 in this instance is used with a single thread of control.

distribution by

$$S(x) \stackrel{D}{=} |2t + tN(x)|,$$

where t is the task index. (All other variates were scaled by t and where necessary an absolute value was taken.) Table 1 shows any small differences that may exist. The task index is either in seconds or in loop iterations. When n is large, the results are dominated by the behaviour determined by $\overline{x}$ with a small remainder arising from the influence of p and σ. It is apparent that KWLarge, KW1, Asymptotic and sample are consistently below or close to the actual job duration while KW2, M&S, and max are upper bounds. The worst case error is about 1s or 1%.

The exponential distribution may occur whenever there is interactive use of the computer. Though we do not normally expect exponentially-distributed task sizes in PPF, the distribution is easily analysed. Table 2 suggests that the metrics are slightly less accurate for an exponential distribution than a normal distribution. The effect may result from a larger s.d., since it can now be seen that estimate KW2, which is proportional to σ^2, diverges. KWLarge and KW1 come near to the job duration and M&S or one of the other metrics are suitable upper bounds.

The Bernoulli distribution, which of course is discrete, does not have an IFR distribution but has finite moments. The Bernoulli distribution may act as a model even for deterministic task duration distributions because periodic system delays may impose a bimodal distribution. Bimodal distributions occur when there are two alternative branches in computer code. Tests were conducted with prob($x = t$), where t is a large delay, being 0.25 and prob($x = t/2$) naturally being 0.75. In all tests, the predictors accurately track the job duration, Figure 10. For small numbers of iterations, the overhead dominates in Figure 10, causing the performance shelf below 4000 iterations. Predictor M&S bounds the result while the two KW predictors are tight lower bounds.

Normal, exponential, Bernoulli and deterministic distributions are common and accurately predicted. Symmetrical distributions are more likely to be applicable to image-processing

applications [14].

7 Simulation Results

A simulation was conducted to verify the large p behaviour of the various estimators. To approach the population statistics 20,000 tasks were distributed. A delay of 0.001 was set and chunk size was one task. Experiments were made at eight processor intervals from 8 to 248 processors. Notice that the model does not account for any congestion introduced through scaling, by analogy with 'hot-spot' contention. Just as on traditional NUMA machines an hierarchical scheme is possible [68], sub-farmers within a large farm may be possible.

Figure 11 shows the estimators for an exponential cdf with parameter $\lambda = 1$ (*i.e.* $\mu = \sigma = 1/\lambda = 1$). The sample estimator and 'M&S' are clearly seen as loose upper bounds. $E[X_{p:p}]$, (6), also tends to act as an upper bound to the simulation results. The double exponential asymptotic estimator is a good estimator for the exponential cdf. Estimator 'KW2' is excluded as it distorts the scaling of the graph, but 'KW1' acts as a tight lower bound. A standard normal distribution was transformed by

$$S \stackrel{D}{=} |2 + N(0,1)|,$$

in order to keep a high proportion of the results before truncating. Negative values are truncated but by rule-of-thumb as 95% of a Gaussian density is found within two standard deviations of the mean the negative values are first pushed accordingly into the positive region. Except for the two upper bound estimators, the sample estimate and 'MS', there is little to distinguish the estimators in practical value, according to the results of the simulation shown in Figure 12. As indicated in Section 3.2, the asymptotic predictor is not to be relied upon for a normal distribution, though in the simulation it was a tighter upper bound than 'M&S' and 'sample'. The simulation for $U(0,1)$ (*i.e.* $\mu = 1/2$, $\sigma^2 = 1/12$) showed that all estimators considered act as upper bounds, Figure 13. This should not be surprising as the uniform cdf is bounded whereas the estimators are primarily intended for unbounded distributions. However, since the uniform distribution is IFR one might have expected the 'KW' estimators

to lie around the simulated value and not to act as an upper bound. Estimators for the Bernoulli cdf, Figure 14, under the same conditions as Section 6.2, $t = 1$, are well-behaved. Despite the fact that the Bernoulli distribution is discrete and bounded, the estimators are all reliable predictors.

8 An Asynchronous Pipeline Estimate

In a synchronous pipeline of farms, the various estimates of maximum latency developed in Section 5 are immediately applicable. If there are asynchronous stages it will be necessary to account for waiting time in a queue before being served at the next stage of the pipeline. With no loss of generality, consider a two-stage asynchronous PPF. The maximum possible latency is made up of the maximum service time at each farm and the maximum waiting time between the two farms. Suppose the pipeline is in steady state and that the task duration distribution in the first farm is exponentially distributed, *i.e.* by Burke's theory its output is also so distributed [69], then delay-cycle analysis [70] can be used to find the waiting-time distribution. In practice, only the Laplace-Stieltjes transform of the waiting-time cdf is available but this easily yields the moments of waiting time which for the second farm (treated as a single server) are given as:

$$E[X] = E[S] + \frac{\lambda E[S^2]}{2(1-\rho)} \tag{19}$$

$$E[X^2] = E[S^2] + \frac{\rho E[S^2]}{1-\rho} + \frac{\lambda E[S^3]}{3(1-\rho)} + \frac{(\lambda E[S^2])^2}{2(1-\rho)^2}, \tag{20}$$

where, as is conventional in queueing theory, λ is the task arrival rate. S is the task duration r.v. If as in conventional notation μ is the task completion rate then $\rho = \lambda/\mu$ is the availability.[11] If the second farm also has an exponential task duration cdf following [71] the single processor results can easily be adapted to the multiple processor case by appropriately setting the task completion rate to $p\mu$.

Figure 15 shows two results from a simulation in which farm two had fifty processors

[11] Equation (19) will be recognized as the Pollaczek-Khintchine equation for M/G/1 queues.

and farm one had either five or forty processors, reflecting differing arrival rates. The task duration rate in farm two was fixed to 3.0 tasks/unit time to prevent saturation. The mean latency is plotted as well as the maximum latency recorded. Using the moment estimates of (19) & (20) and the characteristic maximum of an exponential distribution, (13), the maximum latency over 20,000 tasks is found. Notice that the maximum is from the number of tasks and not the number of processors. The estimates of Section 5 are less successful in finding the asynchronous maximum since they reflect the mode of a normal cdf and not an exponential cdf. In particular, such estimates appear erroneous when the task arrival rate from farm one is low but the task duration in both farms is large. The mean task duration for the first farm was used reflecting (intuitively) the unlikelihood of all three maxima occurring together. The estimate remains an upper bound. If the sampled number of tasks is made lower the estimates lose accuracy. From the large-scale test, it emerged that the mean latency is close to the sum of the two mean service times which implies that system design based on means is likely to be accurate. However, in some conditions the maximum latency departs considerably from the mean which will be an important issue for time-critical systems.

9 Ordering Constraints

On a hypothetically balanced pipeline, an ordering constraint will degrade the latency. With sufficient buffering, no change in throughput occurs because when the order is restored all waiting tasks can be instantaneously output, provided sampling of throughput is at output instances. If sampling of throughput is by a random observer then the output throughput will have the same form of degradation as now discussed for latency. Ordering is a synchronization constraint over the whole of an asynchronous pipeline segment, as the order needs to be restored before leaving the asynchronous segment. Re-ordering can been implemented by forming a linked-list to act as a buffer for out-of-order tasks at the pipeline stage, typically the last stage of an asynchronous segment, where the ordering constraint applies.

Suppose that an asynchronous segment of a pipeline has a capacity for n tasks at the

instant that any one of these tasks becomes ready for processing. Suppose also that (in the worst case) all n tasks start at the same time and their finishing times are distributed as a set of ordered r.v. $\{X_{i:n}\}$, $i = 1, 2, \ldots, n$ with the conditions of Section 3.1. Though, particularly in the case of the extremal value (Section 3.1), the mean may not be the best estimator, nonetheless a practical approach might be to use the mean of the ordered means to estimate the average time that any completing task waits:

$$t_{latency} = \frac{1}{n} \sum_{i=1}^{n} E[X_{i:n}].$$

or if mean latency without a constraint is μ and $E[X_{i:n}] = \mu_i$ then percentage degradation is

$$\left(1 - \frac{n\mu}{\sum_{i=1}^{n}}\right) \times 100\%.$$

The distribution of the means for all order statistics, $i = 1, 2, \ldots, n$ is required. The uniform distribution, $U(0, 1)$, has $E[U_{i:p}] = i/(p+1) = p_i$ which gives a triangular distribution. Unfortunately, other distributions do not have such a convenient formula. Therefore an estimate is proposed. In [53], from $X_{i:p} = F^{-1}(U_{i:p}) = G(U_{i:p})$ a Taylor expansion[12] around p_i is used to approximate $E[X_{i:p}]$ *i.e.*

$$X_{i:p} = G(p_i) + G'(p_i)(U_{i:p} - p_i) + \frac{1}{2}G''(U_{i:p} - p_i)^2 \ldots, \tag{21}$$

where G' denotes $d/duG(u)|_{u=p_i}$, $u = F(x)$. A convenient example is a logistics cdf, $F(y) = (1 + e^{-y})^{-1}$ with $q_i = 1 - p_i = (p - i + 1)/(p + 1)$:

$$\begin{aligned} G(p_i) &= \ln(p_i) - \ln(q_i) \\ G'(p_i) &= 1/p_i - 1/q_i \end{aligned}$$

Taking expectations of both sides of (21), using the result for $E[U_{i:p}]$, gives

$$E[Y_{i:p}] \sim G(p_i) + \frac{p_i q_i}{2(p+2)} G''(p_i) + \frac{p_i q_i}{(n+2)^2} \left[\frac{1}{3}(q_i - p_i)G'''(p_i) + \frac{1}{8}p_i q_i G^{iv}(p_i)\right] \ldots \tag{22}$$

Figure 16 uses expansion (22) to make a continuous estimate of $E[Y_{i:p}]$, i odd. The form of (22) is approximately triangular and other common distributions have the same shape for $E[X_{i:p}]$,

[12] The inverse function relationship is also often used to generate random numbers. Unfortunately, the convergence of the expansion is slow for the extreme order statistic.

bearing in mind that the logistic distribution when suitably parameterized approximates a normal distribution. This result is of course helpful in estimating $t_{latency}$ or $t_{thruput}$.

Measurements have been made on the handwritten postcode recognition application introduced in Section 4 applied on a transputer-based Meiko CS-1. The CS-1 [72] uses proprietary routing hardware and software aside from the normal transputer links. The processing times for complete postcodes was found to have a symmetrical distribution for the preprocessing and classification stages. The final dictionary stage had a Bernoulli distribution as UK postcodes are almost all 6 or 7 characters in length, with the test data mean length being 6.7 characters matched against a complete set for the city of Colchester, UK. Table 4 records the measured percentage throughput degradation, found on a per-stage basis by decoupling each stage with very large buffers. The range, R *cf.* Equation 4, is employed as a measure of variance. Perf. I is the theoretical performance after after ordering degradation, and Perf. II is the degradation due to ordering cumulatively applied.

In fact, there is no need to assemble complete postcodes until the dictionary stage, and additionally using the i860 processors in the Paramid increased performance by an order of magnitude [1]. The dictionary stage has been tested in isolation on the Paramid, since, as remarked in Section 2, the scaling on the Paramid did not allow more than one worker task in the dictionary stage when testing the complete pipeline. The mean time to recognize a postcode of six or seven characters was timed to be 0.027s or 0.13s respectively from a test file of 1945 characters.[13] Message size was a maximum of 112 bytes. The Bernoulli probability for a six character postcode was 0.517. The observed distribution, though closely resembling a Bernoulli distribution, was continuous and bimodal. The processors were arranged in a binary tree topology. Table 5 shows that again all estimates track the results well with small numbers of processors.

[13]The dictionary search employed a trie tree [61] over a small city's postcodes, whereas subsequent implementations have used a syntactic neural net search with an order of magnitude improvement over the complete UK postcode dictionary.

10 Conclusion

PPF is a stylized system development methodology which offers a clear route to the parallelization of an existing sequential application. The application domain is soft real-time continuous-flow embedded systems. PPF employs a linear parallel pipeline. The maximum latency experienced by tasks passing through a PPF application can be found using the same techniques previously applied to the performance of loop-scheduling on traditional NUMA machines. Additional distribution-specific estimates are of value in testing estimates. The pipeline traversal latency experienced by a task maximum is not always best estimated by its mean. Alternatively, the characteristic maximum can be employed to estimate the modal maximum for what may be broadly termed 'exponential' distributions. Fortunately, since the Bernoulli distribution is commonly used to model program loads, the estimate also works for the Bernoulli distribution. Although, on our small-scale system, all predictors except one gave accurate results, inspection of the p-dependency of the equations suggests that predictors based on the global maximum of asymmetrical distributions will underestimate the performance on large numbers of processors. A simulation for a larger number of processors (up to 256) highlighted the effects already seen in the small-scale system. Again, a small sample predictor is rather a loose upper bound as also is one based on the maximum of the maximal distribution. For the distributions most often encountered most predictors have a rough equivalence. This is an important practical finding in view of the profusion of estimators. Another practical point is that if a system is scaled-up the maximum latency will increase according to the prediction equations, that is increased throughput comes at a cost.

Simulated results for asynchronous PPF can be predicted by a combination of delay-cycle analysis and order statistics, though presently the results are confined to exponential cdf. The maximum latency can in some circumstances depart radically from the mean latency. Order statistics also prove helpful in estimating the performance degradation if a task ordering constraint is imposed.

Work has advanced to incorporating our derived results for synchronous pipelines into

a graphical tool for PPF system design. Discrete-event simulation is the chosen vehicle for asynchronous pipeline segments, as we do not have results for general distributions. However, the tool plans to incorporate analytic prediction for the synchronous results using a spreadsheet format in the manner of [73]. A coherent method of combining synchronous and asynchronous pipeline segments, in the presence of feedback loops and folded-back pipelines has been devised.

Acknowledgement

This work was carried out under EPSRC research contract GR/K40277 'Parallel software tools for embedded signal processing applications' as part of the EPSRC Portable Software Tools for Parallel Architectures directed programme.

References

[1] A. Çuhadar, A. C. Downton, and M. Fleury. A structured parallel design for embedded vision systems: A case study. *Microprocessors and Microsystems*, 21:131–141, 1997.

[2] M. N. Edward. Radar signal processing on a fault tolerant transputer array. In T.S Durrani, W.A. Sandham, J.J. Soraghan, and S.M. Forbes, editors, *Applications of Transputers 3*. IOS, Amsterdam, 1991.

[3] S. Glinski and D. Roe. Spoken language recognition on a DSP array processor. *IEEE Transactions on Parallel and Distributed Systems*, 5(7):697–703, 1994.

[4] A-M Cheng. High speed video compression testbed. *IEEE Transactions on Consumer Electronics*, 40(3):538–548, 1994.

[5] D. May, R. Shepherd, and C. Keane. Communicating process architectures: Transputers and occam. In P. Treleaven and M. Vanneschi, editors, *Future Parallel Computers*, pages 35–81. Springer, Berlin, 1987. Lecture Notes in Computer Science #272.

[6] G. V. Wilson. *Practical Parallel Processing*. MIT, Cambridge, MA, 1995.

[7] J. Schaeffer, D. Szafron, G. Lobe, and I. Parsons. The Enterprise model for developing distributed applications. *IEEE Parallel and Distributed Technology*, pages 85–95, August 1993.

[8] A. S. Wagner, H. V. Sreekantaswamy, and S. T. Chanson. Performance models for the processor farm paradigm. *IEEE Transactions on Parallel and Distributed Systems*, 8(5):475–489, May 1997.

[9] S. Pelagatti. *Structured Development of Parallel Programs*. Taylor & Francis, London, 1998.

[10] A. S. Grimshaw, W. T. Strayer, and P. Narayan. Dynamic, object-oriented parallel processing. *IEEE Computer*, pages 33–46, May 1993.

[11] H. T. Kung, L. M. Ruane, and D. W. L. Yen. A two-level pipelined systolic array for convolutions. In H. T. Kung, B. Sproull, and G. Steele, editors, *VLSI Systems and Computations*, pages 255–264. Springer, Berlin, 1981.

[12] A. M. Wallace, G. J. Michaelson, K. G. Waugh, and W. J. Austin. Dynamic control and prototyping of parallel algorithms for intermediate- and high-level vision. *IEEE Computer*, pages 43–53, February 1992.

[13] D. P. Agrawal and R. Jain. A pipeline pseudoparallel system architecture for real-time dynamic scene analysis. *IEEE Transactions on Computers*, 31(10):952–962, October 1982.

[14] S-Y. Lee and J. K. Aggarwal. A system design/scheduling strategy for parallel image processing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(2):194–204, February 1990.

[15] A. Choudhary, B. Narahari, D. M. Nicol, and R. Simha. Optimal processor assignment for a class of pipelined computations. *IEEE Transactions on Parallel and Distributed Systems*, 5(4):439–445, April 1994.

[16] A. Choudhary, W-k. Liao, D. Weiner, P. Varshney, R. Linderman, and M. Linderman. Design, implementation, and evaluation of parallell pipelined STAP on parallel computers. In *IEEE IPPS/SPDP'98*, 1998. Available at `http://dlib.computer.org/conferen/ipps`.

[17] A. C. Downton, R. W. S. Tregidgo, and A. Cuhadar. Top-down structured parallelisation of embedded image processing applications. *IEE Proceedings Part I (Vision, Image and Signal Processing)*, 141(6):431–437, 1994.

[18] I. K. Proudler and J. G. McWirter. Algorithmic engineering in adaptive signal processing: Worked examples. *IEE Proceedings I (Vision, Image, and Signal Processing)*, 141(1):19–26, 1994.

[19] H. Sava, M. Fleury, A. C. Downton, and A. F. Clark. Parallel pipeline implementation of wavelet transforms. In $6^{\underline{th}}$ *International Conference on Image Processing and its Applications*, pages 171–173, 1997.

[20] M. Fleury, A. C. Downton, and A. F. Clark. Karhunen-loève transform: An exercise in simple image-processing parallel pipelines. In *Euro-Par'97*, pages 815–819. Springer, Berlin, 1997. Lecture Notes in Computer Science 1300.

[21] M. Fleury, A. C. Downton, and A. F. Clark. Parallel structure in an integrated speech-recognition network. In *EuroPar'99*, pages 995–1004. Springer, Berlin, 1999. Lecture Notes in Computer Science #1685.

[22] A. C. Downton. Generalised approach to parallelising image sequence coding algorithms. *IEE Proceedings Part I (Vision, Image and Signal Processing)*, 141(6):438–445, 1994.

[23] M. Fleury and A. F. Clark. Parallelizing a set of 2-d frequency transforms in a flexible manner. *IEE Part I (Vision, Image and Signal Processing)*, 145(1):65–72, February 1997.

[24] D. J. Pritchard. Mathematical models of distributed computation. In $7^{\underline{th}}$ *Occam User Group Technical Meeting*. IOS, Amsterdam, 1987.

[25] N. Boden, D. Cohen, R. Felderman, A. Kuwalik, C. Seitz, J. Seizovic, and W. Su. Myrinet: A gigabit-per-second local area network. *IEEE Micro*, 15(1):29–38, 1995.

[26] D. A. Reed and R. M. Fujimoto. *Multicomputer Networks: Message-Based Parallel Processing*. MIT, Cambridge, MA, 1988.

[27] C. E. Leiserson. Fat trees: Universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, 34(10):892–901, October 1985.

[28] L. M. Ni and P. K. McKinley. A survey of wormhole routing techniques in direct networks. *IEEE Computer*, 26(2):62–76, February 1993.

[29] L. G. Valiant. General purpose parallel architectures. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, pages 943–972. Elsevier, Amsterdam, 1990.

[30] J. M. Hill and D. B. Skillicorn. Lessons learned from implementing BSP. In *High Performance Computing and Networking (HPCN'97)*, pages 762–771. Springer, Berlin, April 1997.

[31] R. Beton, S. P. Turner, and C. Upstill. Hybrid architecture paradigms in a radar esm data processing application. *Microprocessors and Microsystems*, 13(3):160–164, April 1989.

[32] C. P. Kruskal and A. Weiss. Allocating independent subtasks on parallel processors. *IEEE Transactions on Software Engineering*, 11(10):1001–1016, October 1985.

[33] Transtech Parallel Systems Group, 20 Thornwood Drive, Ithaca, NY. *The Paramid User's Guide*, 1993.

[34] M. Debbage, M. B. Hill, and D. A. Nicole. The virtual channel router. *Transputer Communications*, 1(1):3–18, August 1993.

[35] D. Durand, T. Montaut, L. Kervella, and W. Jalby. Impact of memory contention on dynamic scheduling on NUMA multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 7(11):1201–1214, November 1996.

[36] G. S. Almasi and A. Gottlieb. *Highly Parallel Computing*. Benjamin/Cummings, Redwood City, CA, $2^{\underline{nd}}$ edition, 1994.

[37] D. E. Culler, J. P. Singh, and A. Gupta. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann, 1999.

[38] S. Hummel and E. Schonberg. Low-overhead scheduling of nested parallelism. *IBM Journal of Research and Development*, 35(5/6):743–765, September/November 1991.

[39] Y. Paek and D. A. Padua. Experimental study of techniques for NUMA machines. In *IEEE IPPS/SPDS'98*, 1998. Available at `http://dlib.computer.org/conferen/ipps`.

[40] M. Dubois and F. A. Briggs. Performance of synchronized iterative processes in multiprocessor systems. *IEEE Transactions on Software Engineering*, 8(4):419–431, July 1988.

[41] G. F. Pfister and V. A. Norton. "hot spot" contention and combining in multistage interconnection networks. *IEEE Transactions on Computers*, 34(10):943–948, 1985.

[42] T. LeBlanc. Shared memory versus message-passing in a tightly-coupled multiprocessor: A case study. In *International Conference on Parallel Processing*, pages 463–464, 1986.

[43] S. S. Lumetta and D. E. Culler. Managing concurrent access for shared memory active messages. In *IEEE IPPS/SPDP'98*, 1998. Available at `http://dlib.computer.org/conferen/ipps`.

[44] R. T. Thomas and W. Crowther. The Uniform system: An approach to runtime support for large scale shared memory parallel processors. In *International Conference on Parallel Processing*, volume 2, pages 245–254, August 1988.

[45] S. Shekhar, S. Ravada, V. Kumar, D. Chubb, and G. Turner. Parallelizing a GIS on a shared address space architecture. *IEEE Computer*, 29(12):42–48, 1996.

[46] H. V. Sreekantaswamy, S. Chanson, and A. Wagner. Performance prediction modelling of multicomputers. In *IEEE* $12^{\underline{th}}$ *International Conference on Distributed Computing Systems*, pages 278–285, 1992.

[47] F. T. Chong, B-H. Lim, R. Bianchini, J. Kubiatowicz, and A. Agarwal. Application performance on the MIT alewife machine. *IEEE Computer*, 29(12):57–64, 1996.

[48] M. Fleury, A. C. Downton, and A. F. Clark. Modelling pipelines for embedded parallel processor system design. *Electronic Letters*, 33(22):1852–1853, October 1997.

[49] CCITT. Draft revision of recommendations H261: video codec for audiovisual services at px64bits/s. *Signal Processing: Image Communication*, 2(2):221–239, 1990.

[50] J. T. Robinson. Some analysis techniques for asynchronous multiprocessor algorithms. *IEEE Transactions on Software Engineering*, 5(1):24–31, January 1979.

[51] E. Gelenbe. *Multiprocessor Performance.* Wiley, Chichester, UK, 1989.

[52] E. J. Gumbel. *The Statistics of Extremes.* Columbia University, New York, 1958.

[53] N. Balakrishnan and A. C. Cohen. *Order Statistics and Inference.* Academic Press, Boston, 1991.

[54] H. O. Hartley and H. A. David. Universal bounds for mean range and extreme observations. *Annals of Mathematical Statistics*, 25:85–99, 1954.

[55] E. J. Gumbel. The maxima of the mean largest value and of the range. *Annals of Mathematical Statistics*, 25:76–84, 1954.

[56] H. Cramér. *Mathematical Methods of Statistics.* Princeton U.P., Princeton, 1946.

[57] T. L. Lai and H. Robbins. Maximally dependent random variables. *Proceedings of the National Academy of Science, USA*, 73(2):286–288, February 1976.

[58] M. G. Kendall and A. Stuart. *The Advanced Theory of Statistics*, volume 1. Griffin, London, UK, $2^{\underline{nd}}$ edition, 1963.

[59] T. L. Lai and H. Robbins. A class of dependent random variables. *Zeitschrift für Wahrscheinlichkeitstheorie*, 42:89–111, 1978.

[60] C. Y. Park and A. C. Shaw. Experiments with a program timing tool based on source-level timing schema. *IEEE Computer*, 24(5):48–57, May 1991.

[61] D. E. Knuth. *The Art of Computer Programming*, volume 2/ Seminumerical Algorithms. Addison-Wesley, Reading, MA, $2^{\underline{nd}}$ edition, 1981.

[62] R. Jain. *The Art of Computer Systems Performance Analysis.* Wiley, New York, 1992.

[63] A. Weiss. *Large Deviations for Performance Analysis.* Chapman & Hall, London, 1995.

[64] S. Madala and J. B. Sinclair. Performance of synchronous parallel algorithms with regular structures. *IEEE Transactions on Parallel and Distributed Systems*, 2(1):105–116, January 1991.

[65] B. W. Weide. Analytical models to explain anomalous behaviour of parallel algorithms. In *International Conference on Parallel Processing*, pages 183–187, 1981.

[66] G. R. Grimmett and D. R. Stirzaker. *Probability and Random Processes.* OUP, Oxford, UK, $2^{\underline{nd}}$ edition, 1992.

[67] M. Fleury, H. Sava, A. C. Downton, and A. F. Clark. Design of a clock synchronization sub-system for parallel embedded systems. *IEE Proceedings Part E (Computers and Digital Techniques)*, 144(2):65–73, March 1997.

[68] S. P. Dandamundi and S. P. Cheng. A hierarchical task organization for shared-memory multiprocessor systems. *IEEE Transactions on Parallel and Distributed Systems*, 6(1):1–6, January 1995.

[69] P. J. B. King. *Computer and Communication Systems Performance Modelling.* Prentice Hall, New York, 1990.

[70] R. W. Conway, W. L. Maxwell, and L. W. Miller. *Theory of Scheduling.* Addison-Wesley, Reading, MA, 1967.

[71] K. Omahen and V. Marathe. Analysis and application of the delay cycle for the M/M/c queueing system. *Journal of the ACM*, 25(2):283–303, April 1978.

[72] A. Trew and G. Wilson. *Past, present, parallel: a survey of available parallel computing systems.* Springer, London, 1991.

[73] S. R. Sarukkai and D. Gannon. SIEVE: A performance debugging environment for parallel programs. *Journal of Parallel and Distributed Computing*, 18:147–168, 1993.

Task Index	Job Duration	Prediction (s) KWLarge	KW1	KW2	M&S	Sample	Asymptotic	max
2.0	106.2368	105.5765	105.5765	107.7293	107.7293	106.5473	105.6076	107.3153
1.0	53.2737	53.1466	52.9386	54.0188	54.0188	53.4247	52.9550	53.8086
0.5	26.8061	26.7416	26.6367	27.0751	27.1811	26.8807	26.6458	27.0726
0.25	13.5591	13.4763	13.4763	13.6926	13.7523	13.5990	13.4815	13.6951
0.125	6.9443	6.9239	6.8966	7.0014	7.0385	6.95876	6.9000	7.0068
0.625	3.6446	3.6298	3.6154	3.6655	3.6903	3.64734	3.6177	3.6716
0.0313	1.9839	1.9780	1.9701	1.9932	2.0115	1.9869	1.9719	1.9992
0.0156	1.1622	1.1563	1.1516	1.6260	1.1761	1.1610	1.1530	1.1676
0.0078	0.7358	0.7344	0.7315	0.7376	0.7472	0.7372	0.7325	0.7411
10k	1.0646	1.0620	1.0562	1.0731	1.0778	1.0655	1.0565	1.0729
9k	0.9646	0.9596	0.9561	0.9713	0.9756	0.9645	0.9564	0.9711
8k	0.8622	0.8581	0.8549	0.8684	0.8723	0.8624	0.8552	0.8683
7k	0.7602	0.7567	0.7539	0.7657	0.7692	0.7605	0.7542	0.7656
6k	0.6616	0.6581	0.6556	0.6657	0.6690	0.6613	0.6559	0.6657
5k	0.5602	0.5568	0.5548	0.5632	0.5661	0.5595	0.5560	0.5631
4k	0.4631	0.4598	0.4582	0.4649	0.4675	0.2645	0.4584	0.4649
3k	0.3652	0.3628	0.3616	0.3666	0.3690	0.3643	0.3618	0.3667
2k	0.2777	0.2754	0.2745	0.2778	0.2801	0.2764	0.2746	0.2779
1k	0.2348	0.2332	0.2328	0.2343	0.2375	0.2338	0.2329	0.2345

Table 1: Predictions for a normal Distribution

Task Index	Job Duration	Prediction (s) KWLarge	KW2	M&S	Sample	Asymptotic	m_p
2.0	46.9520	48.2911	55.4189	48.7672	48.8607	47.8005	47.3529
1.0	23.6496	24.3126	27.8468	24.5557	24.5990	24.0690	23.8421
0.5	11.9928	12.3234	14.0629	12.4499	12.4682	12.2032	12.0867
0.25	6.1452	6.3254	7.1672	6.3935	6.3993	6.2669	6.2057
0.125	3.2466	3.3353	3.7274	3.3746	3.3738	3.3078	3.2740
0.0625	1.8060	1.8505	2.0231	1.8755	1.8713	1.1838	1.8182
0.0313	1.0899	1.1060	1.1729	1.1237	1.1177	1.1015	1.0880
0.0156	0.7594	0.7617	0.7804	0.7764	0.7688	0.7611	0.7503
0.0078	0.6060	0.7617	0.5987	0.6115	0.6003	0.5995	0.5887
10k	0.5002	0.5131	0.5807	0.5187	0.5186	0.5084	0.5040
9k	0.4564	0.4678	0.5285	0.4730	0.4728	0.4636	0.4597
8k	0.4142	0.4247	0.4785	0.4295	0.4292	0.4210	0.4175
7k	0.3709	0.3796	0.4265	0.3839	0.3835	0.3763	0.3732
6k	0.3327	0.3401	0.3801	0.3441	0.3434	0.3373	0.3346
5k	0.2969	0.3018	0.3349	0.3055	0.3046	0.2995	0.2972
4k	0.2626	0.2675	0.2935	0.2710	0.2697	0.2657	0.2638
3k	0.2413	0.2446	0.2639	0.2481	0.2463	0.2432	0.2418
2k	0.2337	0.2342	0.2466	0.2380	0.2353	0.2333	0.2323
1k	0.2337	0.2783	0.2908	0.2830	0.2795	0.2775	0.2765

Table 2: Predictions for an Exponential Distribution

Execution sequence	Function name	Per frame exec. time (normalized)
1	clear_picture	0.147
2	copy_field	0.196
3	read_picture_frame	1.200
4	copy_macro_block_data	0.728
5	encoder_motion_estimation_h261	24.306
6	encoder_decisions_h261	6.392
7	forward_transform_picture	2.230
8	h261_forward_quantize_picture	1.398
9	tm_forward_scan_picture	0.632
10	forward_run_level_picture	0.712
11	h261_code_picture	0.591
12	inverse_run_level_picture	0.486
13	tm_inverse_scan_change_picture	0.634
14	h261_inverse_quantize_picture	0.991
15	inverse_transform_picture	2.232
16	reconstruct_h261	2.813
17	macro_block_to_line	0.531
18	tm_calculate_snr	0.753
19	tm_display_statistics	0.134
20	write_picture_file	1.217

Table 3: Profile Statistics for the H.261 encoder

Stage	Mean (μ) Proc. time (ms)	Min. time (ms)	Max. time (ms)	$R/2$	$\frac{\mu+(R/2)}{\mu}$	% degrad.	Perf. I (pcodes/s)	Perf. II (pcode/s)
PreProc.	487	406	698	146	1.30	17	9.1	9.1
Class.	773	651	1060	205	1.30	18	9.0	7.5
Dict.	2802	629	5349	2360	1.84	36	7.0	4.8

Table 4: The Effect of Ordering Constraints on a Handwritten Postcode Recognition PPF

Number of processors	Actual time (s)	Estimate (s) KWLarge	KW1	KW2	MS	Sample
8	2.88	2.99	3.07	3.28	2.98	3.09
7	3.30	3.36	3.44	3.59	3.34	3.44
3	7.61	7.34	7.44	7.33	7.25	7.33

Table 5: Estimates for the Dictionary Stage of a Handwritten Postcode Recognition PPF

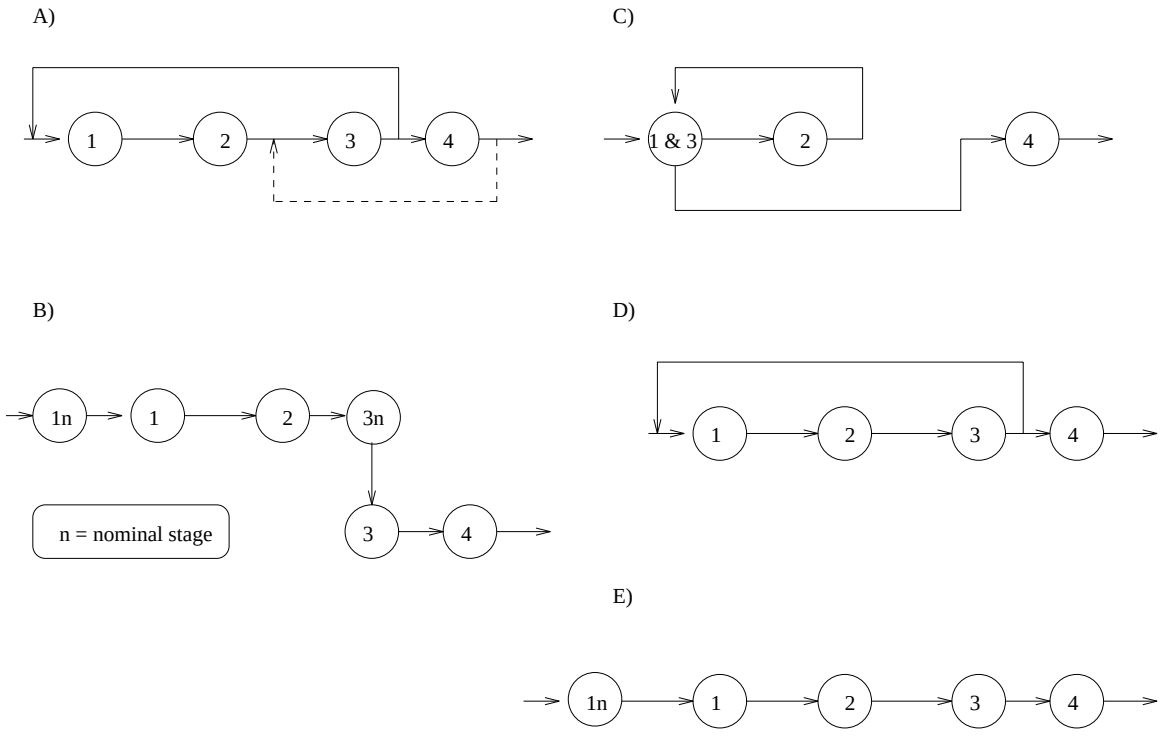

Figure 1: A) Two simple feedback loops B) Replacement by nominal stages C) Folded-back pipeline D) After unwrapping E) After substituting a nominal stage.

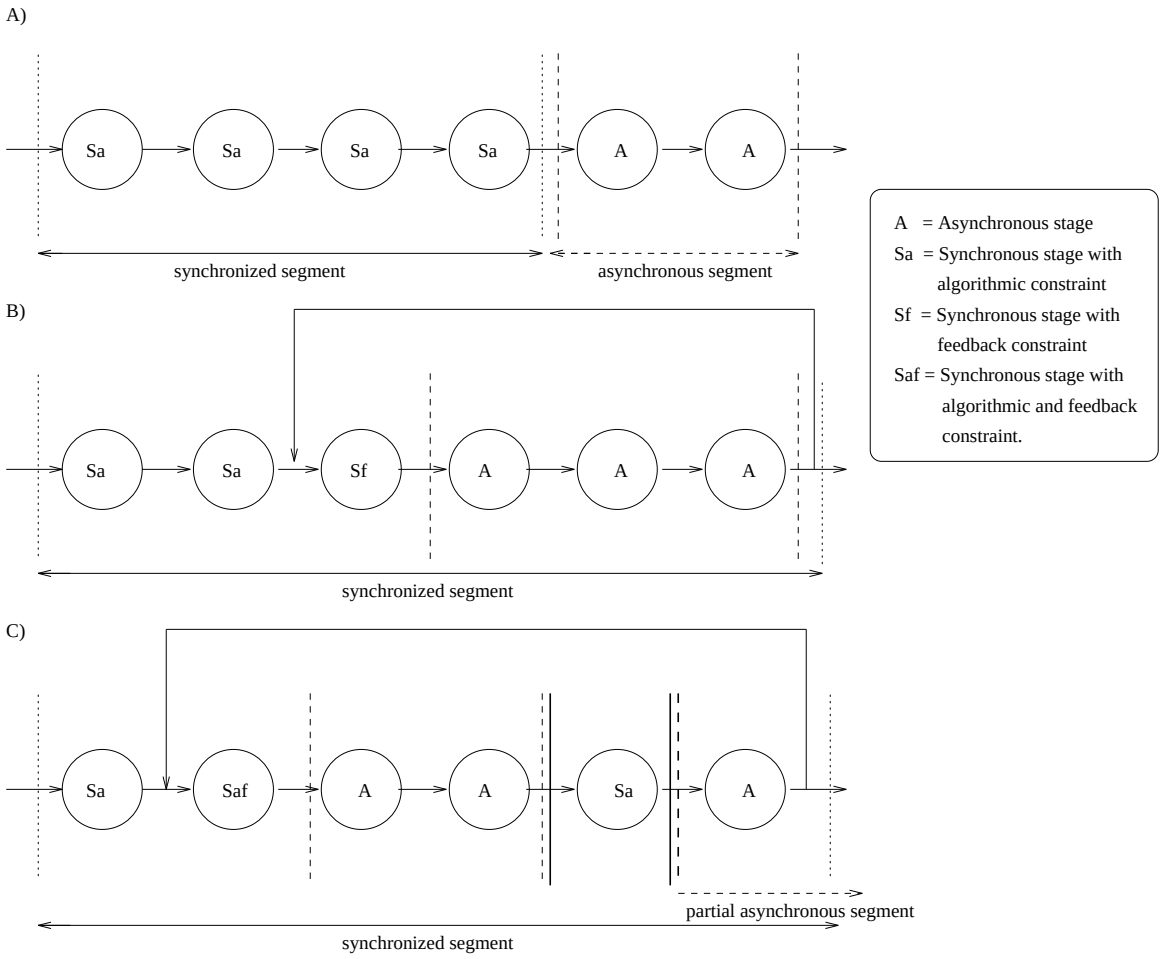

Figure 2: Pipeline splittings: A) Disjoint segments B) Singly Nested segments C) Multiply Nested Segments

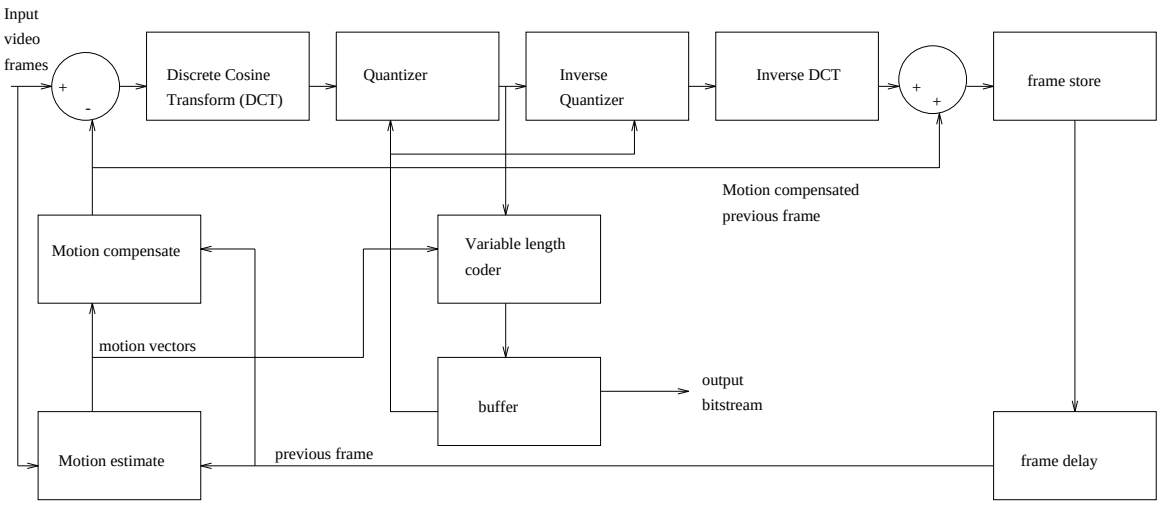

Figure 3: Block Diagram of the Structure of the H.261 Encoder

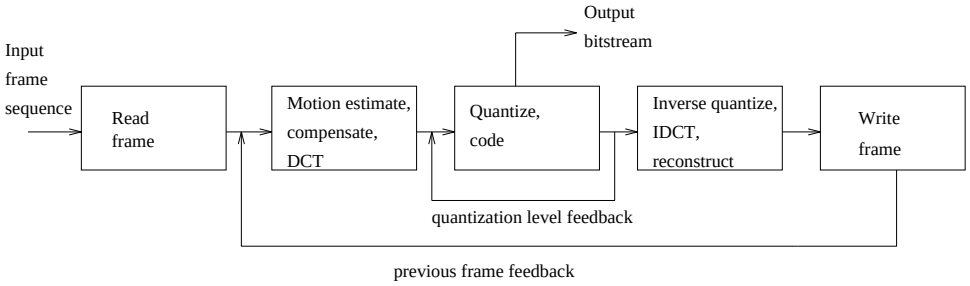

Figure 4: Simplified Linear Pipeline Architecture for the H.261 Encoder

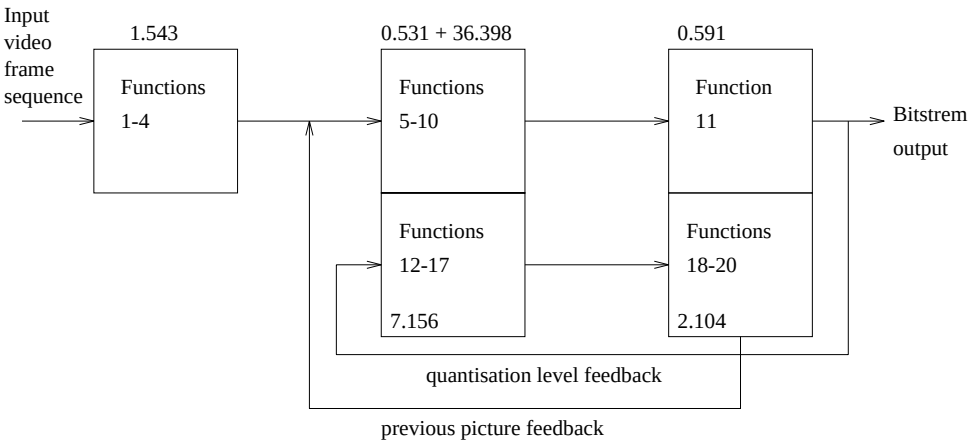

Figure 5: Folded pipeline H.261 Encoder. Refer to Table 3 for function number key and timings.

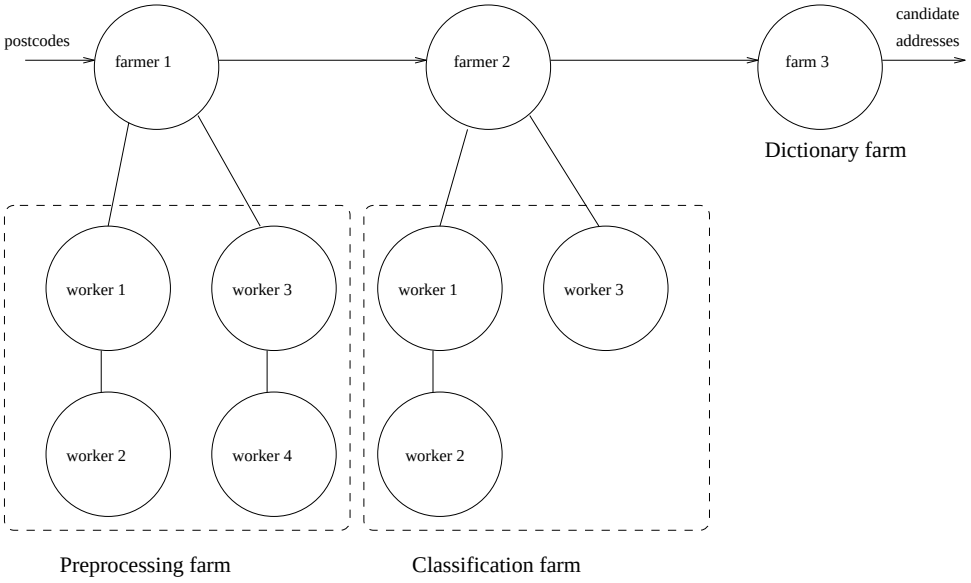

Figure 6: Folded pipeline H.261 Encoder

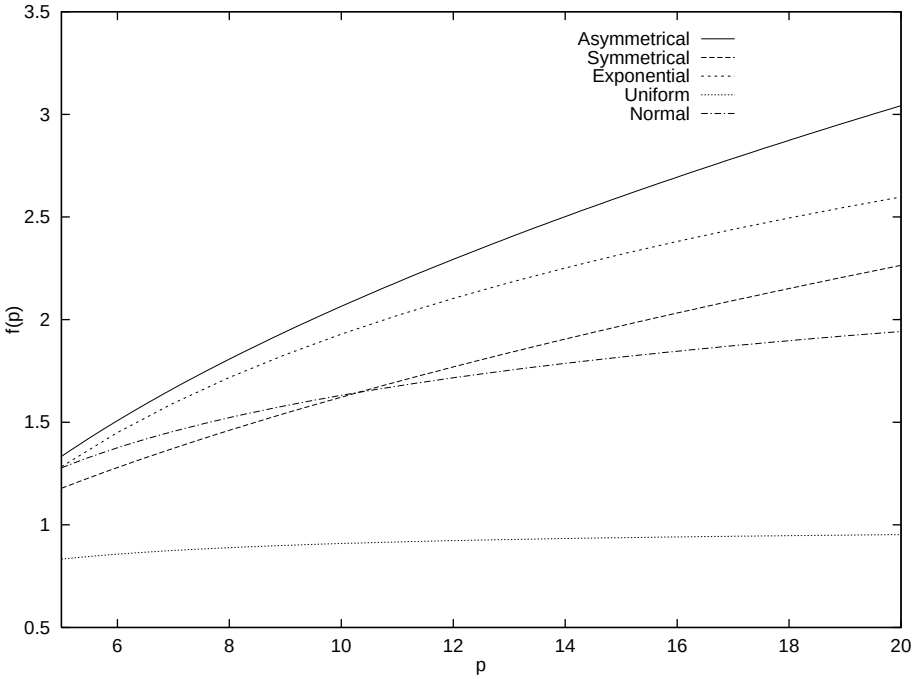

Figure 7: The Mean of the Maximum for Various Distributions

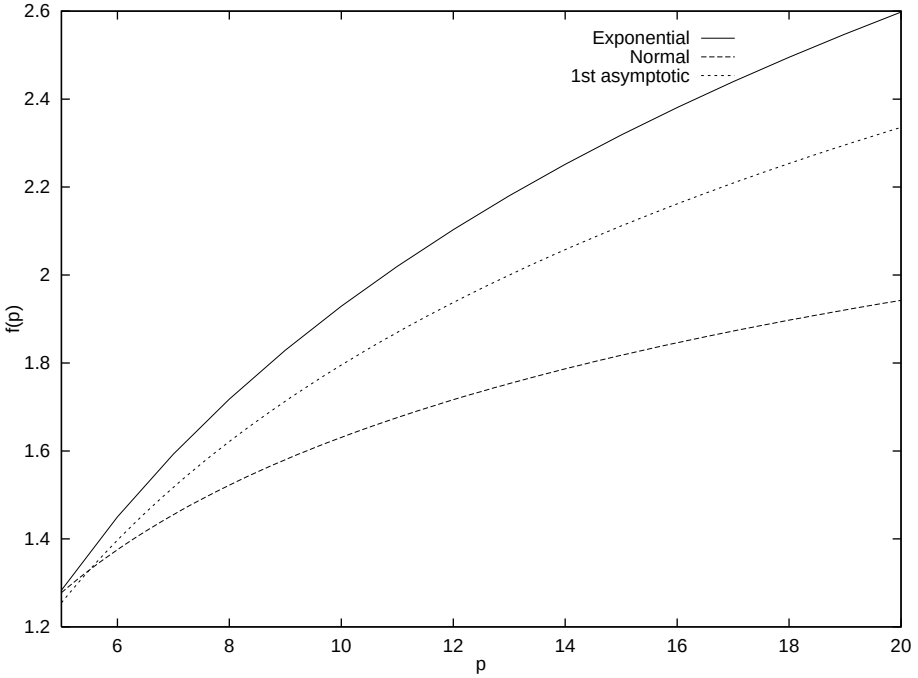

Figure 8: Behaviour of the Asymptotic Distribution's Mean of the Maximum

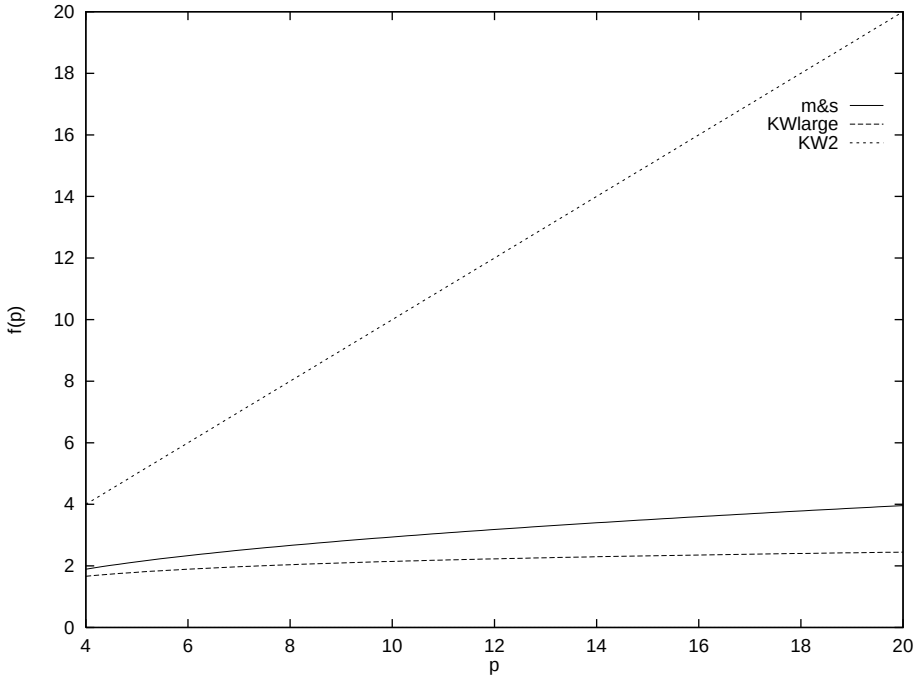

Figure 9: Normalized Prediction Equations

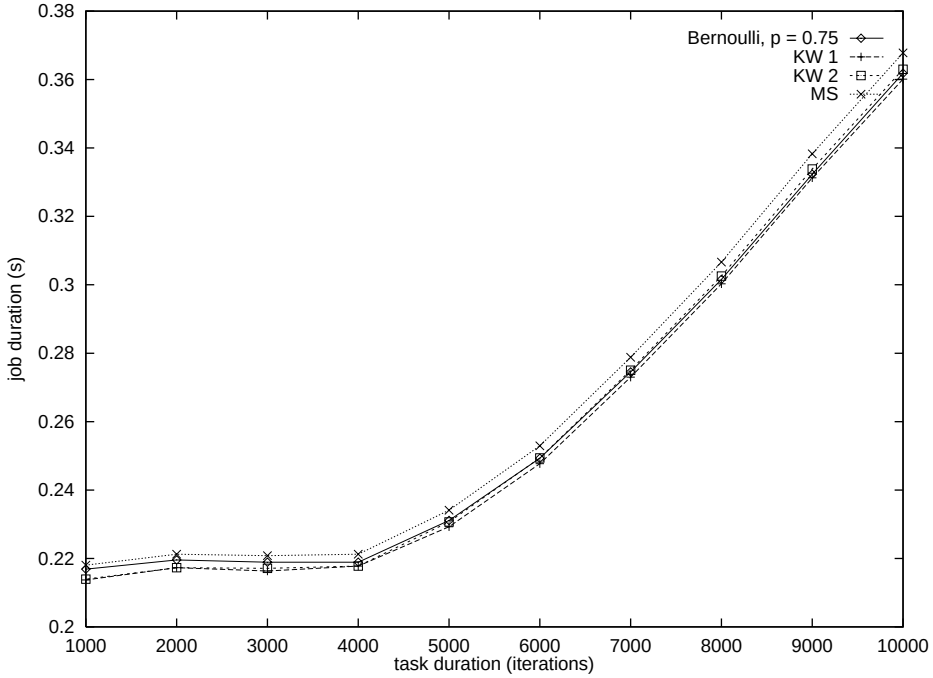

Figure 10: Small Duration Bernoulli Distribution

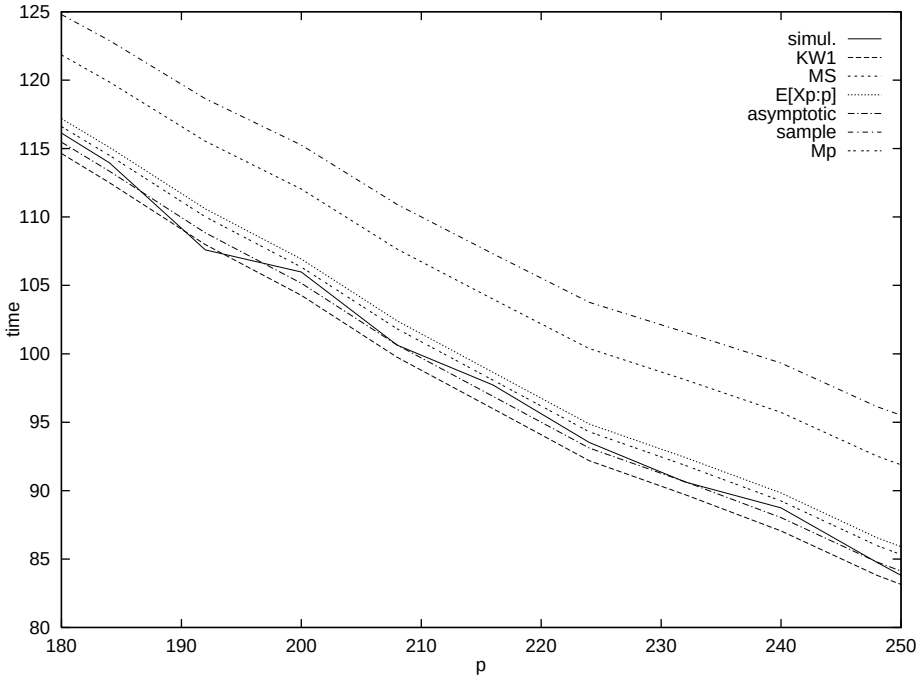

Figure 11: Simulation of Exponential Distribution

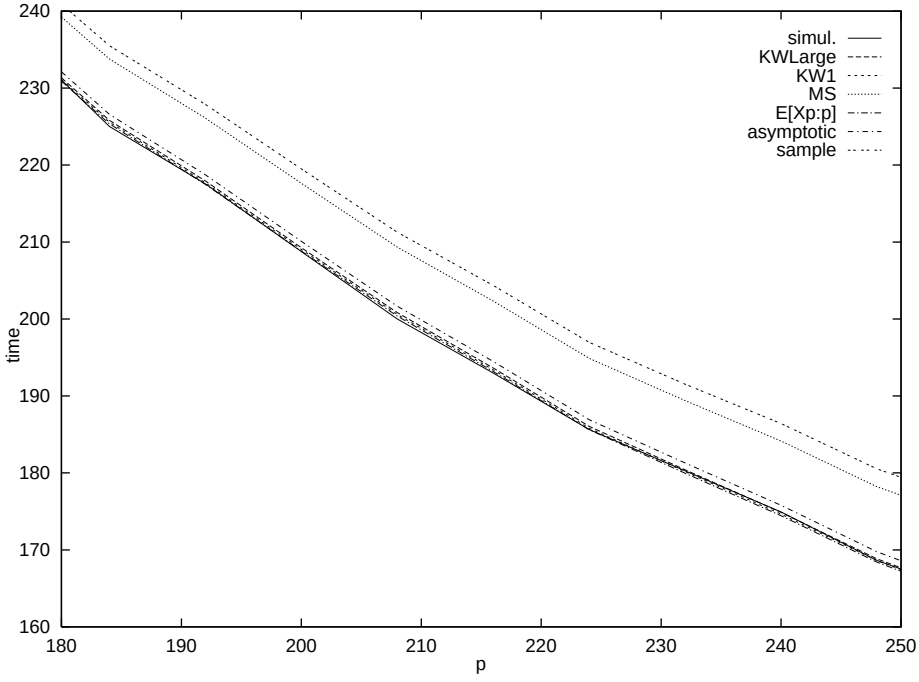

Figure 12: Simulation of Normal Distribution

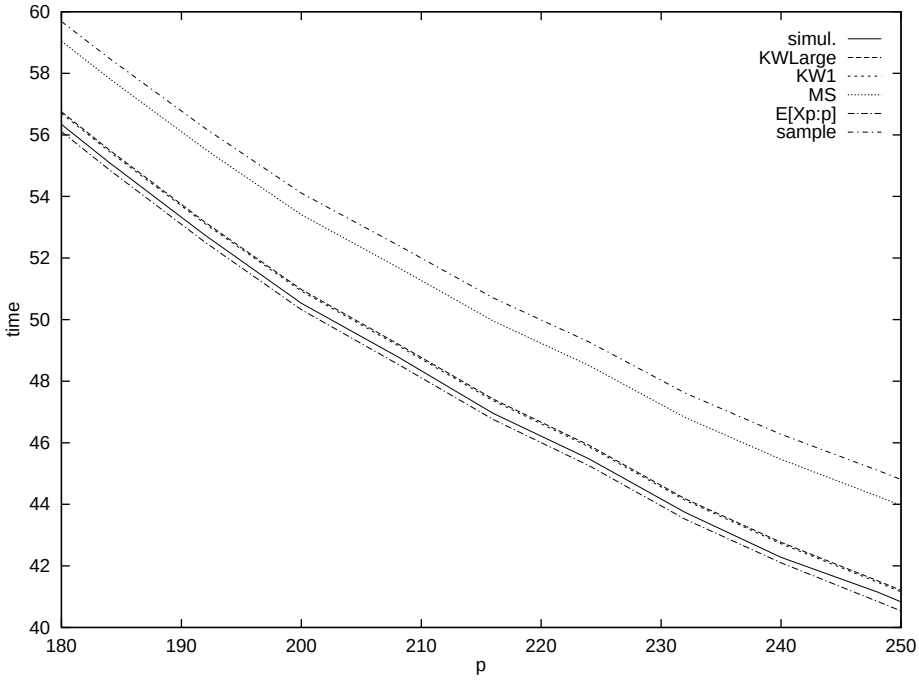

Figure 13: Simulation of Uniform Distribution

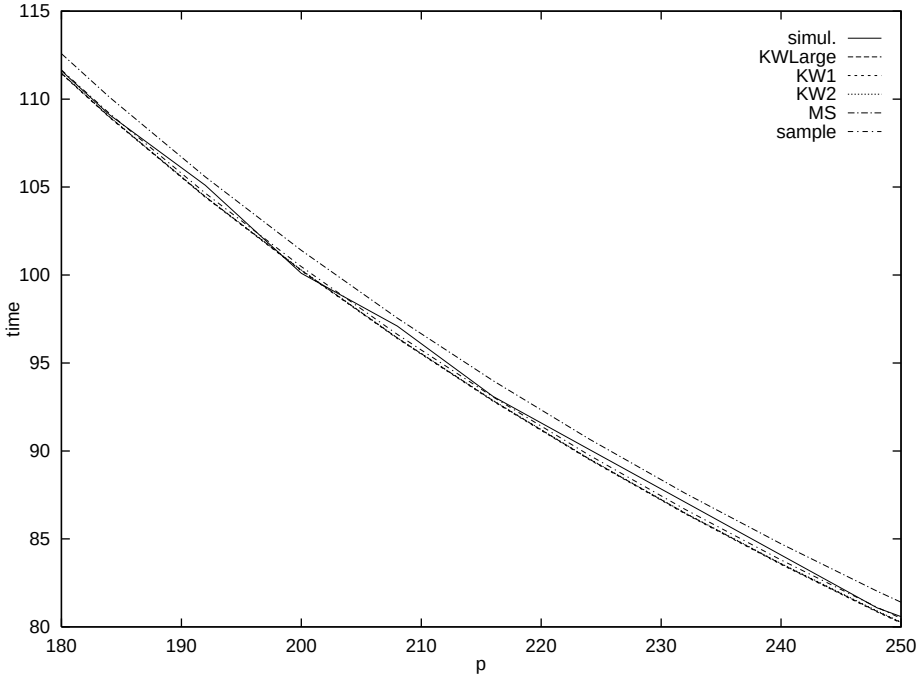

Figure 14: Simulation of Bernoulli Distribution

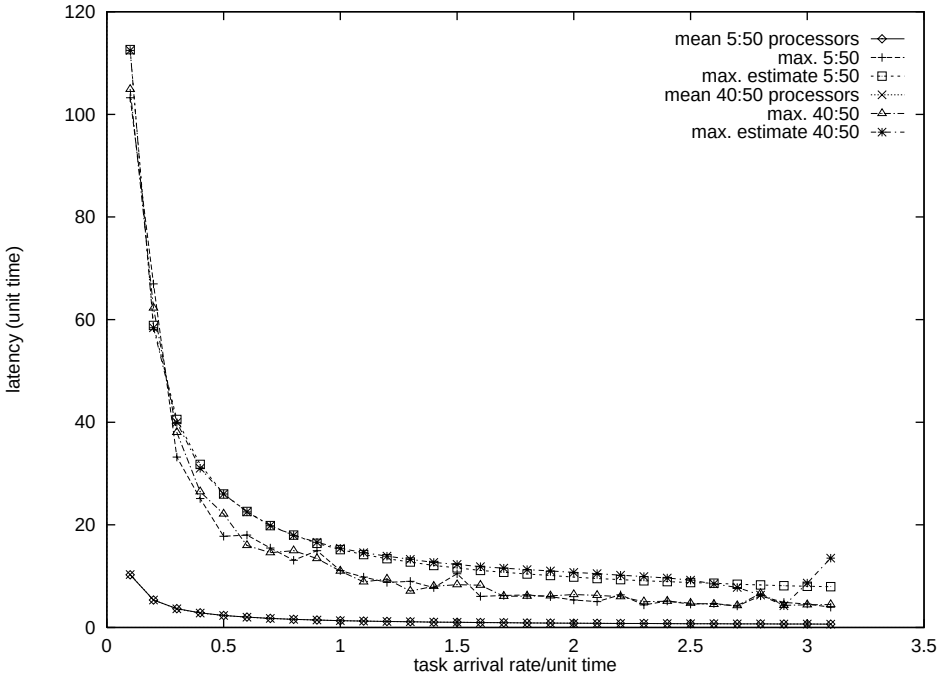

Figure 15: Maximum Latency for a Two-Stage Asynchronous PPF

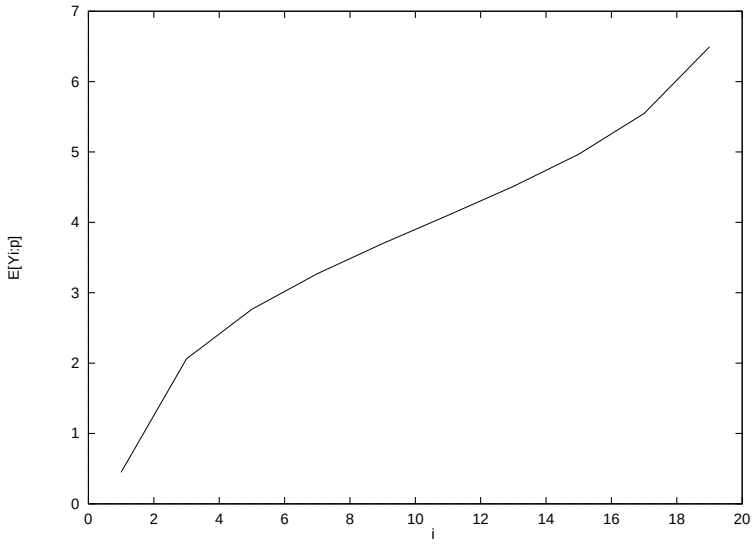

Figure 16: Approximate mean order statistics for the Logistics distribution