Scheduling Schemes for Data Farming

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

Department of Electronic Systems Engineering, University of Essex,

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

tel: +44 - 1206 - 872795

fax: +44 - 1206 - 872900

e-mail fleum@essex.ac.uk

Abstract

The use of order statistics to arrive at a scheduling regime is shown to be applicable to data farms running on second-generation parallel processors. Uniform and decreasing task-size scheduling regimes are examined. Experimental timings and a further simulation for large-scale effects were used to exercise the scheduling regimes. The paper also considers a number of other scheduling schemes for data farms. It is shown that a method previously used for loop-scheduling is preferable, particularly as a form of automatic and generalised scheduling for data farming where there is a data-dependent workload.

1 Introduction

A processor or data farm [1] 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 commonly-used paradigm in parallel processing [2] and appears in numerous guises: some (network-of-workstations) NOW-based [3]; some based on dedicated multicomputers [4]; some constructed as algorithmic skeletons [5]; and some implicitly invoked in parallel extensions of C++ [6]. A staple scheduling method for data-farm tasks is demand-based farming which gives the potential for automatic, run-time load balancing. A missing parameter in the data

farm design is the size of task that is farmed out. Clearly, changing the task size will affect performance. This paper proposes that on current parallel systems a statistical approach is capable of supplying the missing parameter. Choosing the task size at run-time is a form of dynamic scheduling.

In the class of applications of interest, tasks can frequently be broken into jobs. We define the duration of a test run to be the time to complete a set of tasks. For example, we have parallelized handwritten postcode recognition [7], with characters (jobs) within a postcode (task) from a file of postcode images (run); a hybrid video encoder [8], with blocks (jobs) of a video row (task) taken from a sequence of frames (run); and 2-D frequency transforms [9], with rows (jobs) in a sub-image (task) from a batch of images (run). The methods considered in this paper assume that there exists some knowledge of the service time distribution of the jobs within a task. This knowledge is arrived at by repeated timing runs of real-world test data processed by sequential development code which will give an estimate of the statistical moments. The distribution or its symmetry can be estimated by inspection of the job service time histogram. A more precise estimator is a chi-square test on binned data, or through the Kolmogorov-Smirnov test for continuous data [10] if an initial hypothesis for the distribution exists. Where the distribution of job computation times is not known in advance then demand-based farming with equal data for each task remains appropriate, for example in the computation of radiosity in graphics [11]. The optimal task granularity is then problematic and therefore computation should be coarse-grained to ensure that communication can be overlapped with computation. This class of application is not covered in the paper. The same applies to distributed systems with a varying background load, when alternative distributed methods of load-balancing may be appropriate [12].

In a detailed study [4], data farming was applied with constant task durations as is common in numerical analysis applications. In the study, 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. The methods considered

in this paper do not normally apply to store-and-forward networks, but we do not expect this to be too restrictive with current parallel and distributed systems.

In first-generation parallel systems, topology was an important issue [13]. With the advent of generalized communication methods such as the routing switch [14], the 'fat' tree [15], and 'wormhole' routing [16] the situation appears less critical. On second-generation machines, wormhole communication has reduced the message set-up component. Routing congestion can be alleviated either by a randomized routing step [17], or by a changing pattern of communication generated through a latin square [18]. For predicting the overall time of a run involving a large number of task-bearing messages, the mean latency will be sufficient provided the distribution of transfer times is infinitely divisible[1], as is the case for deterministic (constant), normal, or exponential distributions. The advantage for the farm developer were this to be the case is that performance can be predicted from the distribution of task component times alone once the mean communication latency is established for a particular machine.

This paper aims to show that statistically-based methods of scheduling new to data farms do approach optimal run times. The rest of the paper is organized as follows. Previous scheduling schemes for data farms are considered in Section 2. Section 3 outlines the basic prediction equations encountered. Section 4 analyses proposed uniform and decreasing task-size scheduling schemes. The analytical results are compared with experimental timing results and a simulation in Section 5. Finally, Section 6 describes future work and Section 7 draws some conclusions.

[1] z has an infinitely divisible distribution if, for each positive integer n, there are independent and identically distributed (i.i.d.) random variables $\{z_1^n, z_2^n, \ldots, z_n^n\}$ such that $z = z_1^n + \cdots + z_n^n$. If tasks do not interfere in a non-linear fashion then it can be proved that the effect of communication overhead on a per-job basis can be subsumed in a linear way into the processing time for a task [19, p. 252] so that the equations in Section 4 hold.

2 Existing scheduling schemes for data farming

To some extent a scheduling system is a compromise between generality and performance. Global optimization, which is a form of static scheduling, is at the performance end of the spectrum. Because global optimisation is an NP-complete problem heuristics [20] are necessary, though all pertinent system characteristics should be taken into account. In some embedded systems, where the work-load is deterministic, for example in repeatedly performing a Fast Fourier Transform, global optimisation has a place within a data-farming regime. As optimization is time consuming, off-line partitioning of the work is usually required.

In contrast, demand-based farming is a generalised method of scheduling. In demand-based farming, for which the number of tasks should greatly exceed the number of processes, returning processed work (or a suitable token) forms the basis of a request for more work from a central 'farmer'. The computation time of an individual task should exceed the maximum two-way data-transfer time from worker processor to farmer processor so as to hide communication latency. Initially, the central farmer loads all processors with some work, which may be buffered locally, i.e. a static-scheduling phase. To avoid subsequent work starvation on some processes, no worker processor should be able to complete initial processing before the loading phase is over. Subsequently, after making a request, the worker should be able to process locally-buffered work while its request is answered.

Linear programming has been used to predict a lower bound on the performance of data farms [21], i.e. the performance beyond which it is not possible to improve due to the physical bounds of the system. The prediction assumes that store-and-forward communication is employed. A further assumption was that task duration times are deterministic. However in [4], varying task durations were represented by mean values but the linear-programming technique was retained. Maximum performance was more closely estimated by including a number of overhead terms. In one version of the scheme [22], tasks were placed on workers in advance as if the distribution of tasks had been formed by demand-based farming. In theory, task granularity can be found by optimising the governing equations for the model.

In many cases, reported timings on unoptimised regimes are accurate to 5%. However, the model apparently is restricted to transputer systems which are now effectively obsolete.

Scheduling schemes based on a fixed-size communication overhead have been applied to Non-Uniform Memory Access (NUMA) multiprocessors in what resembles a version of data-farming for fine-grained computation. On NUMA machines a task queue is commonly kept. One machine will perform sequential phases of an algorithm while independent loop iterations (identified by a parallelizing compiler) are scheduled between available processes. Alternatively, where algorithmic and/or system perturbation is found, rather than employing static scheduling at compile time, scheduling at run-time [23, 24] is a means of smoothing out the perturbations. A recent study to ascertain the optimal scheduling policy on NUMA machines [25] using a statistical approach prompted the present enquiry into the optimal scheduling scheme for data farming.

3 Background equations

Order statistics form the basis of the task-scheduling schemes under consideration. The size of each task is varied by changing the number of jobs making up that task. All jobs forming tasks within a particular run are assumed to have the same job service time distribution. From order statistics, equations for the run completion time are derived which depend on the task size. By minimizing a run completion-time equation, the optimal task size is found (Section 4.1). In Section 4.2, order statistics are also applied to decreasing task sizes. It may seem strange 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 [26], because of the changing statistical properties when adding (convolving) distributions.

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

∞.[2] From

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

or otherwise the pdf of the maximum order statistic [27] is derived as

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

with $F(\cdot)$ the cumulative distribution function (cdf) and $F_{i:p}(\cdot)$ the cdf of the i^{th} order statistic where $i = 1, 2, \ldots p$. Therefore, applying the expectation operator, the mean of the maximum is

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

Provided that the mean, μ, and the standard deviation (s.d.), σ, of the common cdf exist, it is possible to use the calculus of variations[3] to find the maximum of μ_p across all distributions [29] (with region of support from 0 to 1):

$$\max(\mu_p) = \mu + \sigma \frac{p-1}{\sqrt{2p-1}}, \tag{4}$$

If there was a perfect parallel decomposition, with p now the number of processors, then an upper bound to the run time would be

$$d = \frac{n\mu}{p} + \frac{nh}{pk}, \tag{5}$$

where p is the number of processors in a farm, n is the total number of jobs and k is the number of jobs in a task. h is a constant per task overhead, which includes the cost of communication and any central overhead.

Because of the communication distribution condition mentioned in Section 1, for the purposes of estimating the overall run time, h can be a mean. This point is worth emphasizing: if second-generation multicomputers no longer have non-linear message communication time a second-order statistic is representative and this form of estimate can be applied.

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

[3] There is an alternative derivation using the Cauchy-Schwartz inequality [28].

Theoretical discussion concerns what is the correct way to find the remaining time beyond d to arrive at a statistically correct estimator. In Section 5, this problem is considered empirically.

In a paper by Madala and Sinclair [30], a distribution-free upper bound to performance was proposed based on (4). When the first processor becomes idle there are naturally $p-1$ active processors. The mean remaining time of the run is

$$E[R_{ms}] = k\mu + \sigma\sqrt{\frac{k(p-2)^2}{2p-3}} + h, \tag{6}$$

i.e. as if the first processor finishes just when $p-1$ tasks are assigned but have not been transferred to the remaining processors. Therefore the mean total time is bounded by

$$E[T_{ms}] \leq d + E[R_{ms}]. \tag{7}$$

An alternative set of prediction equations is based on the characteristic maximum, which is an average that arises in a natural way for extremal statistics. Define m_p such that for a cdf H

$$p(1 - H(m_p)) = 1, \tag{8}$$

i.e. out of p timings just one is greater than m_p, the characteristic maximum. In work by Kruskal and Weiss [31], several bounds arose based on finding m_p for different task-size regimes and predicated on increasing failure rate (IFR) distributions [32]. The IFR restriction is not burdensome as many common distributions are IFR, e.g. truncated normal, Weibull (rate ≥ 1), gamma with coefficient of variation (c.o.f.) ≥ 1, exponential, uniform and deterministic. Heavy-tailed distributions such as the Cauchy and Pareto distributions, which are uncommon in computing, are not suitable for these methods of scheduling. m_p is taken to be the time to finish after the last task has been dispatched for processing. Notice that the s.d. of k tasks with common distribution is $\sigma\sqrt{k}$.

We consider just one result from [31]. An easily-derived upper bound for the time up to when the last task is taken is

$$E[T_{start}] \leq \frac{n-kp}{p}\left(\mu + \frac{h}{k}\right). \tag{9}$$

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

$$m_p \approx k\mu + \sigma\sqrt{2k\ln p} \tag{10}$$

$$E[T_{KW_{large}}] \approx d + \sigma\sqrt{2k\ln p}. \tag{11}$$

Equation (11) is derived via the theory of large deviations, see Appendix A, though it is closely related to extremal estimators for the normal distribution [33].

4 Task scheduling

4.1 Uniform task size

An optimal value of k can be found by equating the derivative w.r.t. k to zero, though strictly this optimises the bounding equation and not necessarily the task size. Unfortunately, the resulting expression may not give a closed-form solution, for example from (7)

$$\frac{p\mu k_{opt}^2}{n} - \frac{\sigma k_{opt}^{3/2} p(p-2)}{\sqrt{2p-3}n} = h. \tag{12}$$

In NUMA applications, it is possible that the first process to reach the beginning of the loop will calculate the optimal task size in which case (12) would require a costly numerical calculation. However, notice that for embedded applications it may be possible to pre-calculate k_{opt}.

From (11) the optimal value of k is

$$k_{opt_{kw}} = \left(\frac{\sqrt{2}nh}{\sigma p\sqrt{\ln(p)}}\right)^{2/3}, \tag{13}$$

which can be pre-calculated. Equation (13) is independent of μ and therefore is likely to be unreliable, even though $k_{opt_{kw}} \propto h/\sigma$ is intuitively correct. A prediction for the optimal time based on substituting back $k_{opt_{kw}}$ is

$$t_{min_{kw}} \approx \frac{n\mu}{p} + 2.38\left(\frac{\sigma^2 nh\ln p}{p}\right)^{1/3}. \tag{14}$$

4.2 Decreasing task size

Suppose that there are $i = 1, 2, \ldots, v$ rounds of task distribution but that at each round tasks are dispatched in a batch of p_i sets of k_i tasks then from (4)

$$E[X_{p_i:p_i}] \approx \mu k_i + \sigma\sqrt{k_i p_i / 2}, \tag{15}$$

is a bound on the mean time for all processors, p, to complete for all distributions in which the first two moments exist. In the worst case, just one processor will exceed μk_i. For conservative scheduling, enough work must be still available to keep the remaining processors active in most circumstances. The same considerations recursively apply until the predicted maximum duration for a task approaches the overhead time for sending a task. Fig. 1 illustrates how the total run-time might approach $\mu n/p$ if the maximum departure from the mean old task duration is the same as the new task duration at each scheduling round, with $p_1 = p, p_{i\neq 1} = p - 1$. The idle time experienced by $p - 1$ processors is that time at the end of the final round while the last processor finishes.

Suppose

$$k_i = \frac{n}{p y_i}, \tag{16}$$

with fixed $y_i = 2$. With k_i so set the task size decreases exponentially and half the work is statically allocated in the first round, which is the 'Factoring' regime [34]. Assume that the finishing times at the first scheduling round are distributed as $E[X_{i:p}]$, $i = 1, 2, \ldots, p$, with the r.v. X being the execution times. An approximate solution to setting y_i, thus restricting the idle time in the general case, is shown in Appendix B.

4.3 Heuristic scheduling schemes

In [20], for safe self-scheduling (SSS), in effect a variant of Factoring, it is shown that if the task duration cdf is modelled as Bernoulli a value of $y_i \leq 2$ will limit the idle-time to $pt_{max}/(p-1)$, where t_{max} is the maximum time of any task.[4]

[4]The Bernoulli distribution models long and short conditional branches (i.e. Binomial cdf, $B(1, q)$, where q is the probability of a long branch). For example in the postcode recognition application mentioned in

In [35], a similar limit to the idle time, guided self-scheduling (GSS), was proposed but the rate of task-size descent, which is $(1 - 1/p)^i$, appears too steep.[5] [36] show that for a monotonically decreasing task size GSS will behave badly and propose a linear rate of descent, 'trapezoidal self-scheduling' (TSS). However, in [20], a large-scale test showed that TSS performed weakly if the task cdf was a Bernoulli distribution.

4.4 Discussion

For general distributions, a statistical approach appears necessary rather than heuristics. In theory, it is possible to examine the distribution of all order statistics to find how task finishing times are distributed in the mean. The optimal general scheduling scheme may employ this information in order to minimise the idling time spent by processors as they wait for the last tasks to complete in a run, a conservative scheduling regime.

For a uniform distribution, $U(0,1)$, $\{E[U_{i:p}] = i/(p+1) = p_i\}$ form a triangular distribution of finishing times (and hence a triangular distribution of idling times). Other distributions do not have such a convenient formula. In [37, 27], from $X_{i:p} = F^{-1}(U_{i:p})$ a Taylor expansion around p_i approximates $E[X_{i:p}]$. In Fig. 2, this expansion for a logistic distribution has been used to plot a continuous estimate of $E[X_{i:p}]$, i odd, Again, the idling times roughly have a triangular distribution. In practical situations, symmetrical distributions are likely to have the same balanced ratio of idle time to active time (since the logistic distribution when suitably parameterized can model the ubiquitous normal distribution). It follows that when using decreasing task-size scheduling at most half the available work needs to be reserved (for symmetrical distributions). The Factoring method is predicated on a triangular distribution of job times and is therefore suitable for all symmetrical distributions.

Section 1 [7], UK postcodes can be six or seven characters.

[5] After p rounds, $(1 - 1/p)^p \to 1/e \approx 1/3$, which is approximately the equivalent of $y_i = 3/2$ in (16).

5 Scheduling results

5.1 Timings

In our tests on an eight module multicomputer, the Transtech Paramid [38], application tasks were run on a superscalar i860 in single-threaded mode at 50 MHz, with all communications handled by a transputer coprocessor. The raw bandwidth of a transputer link is 20 Mbit/s. Though the Paramid employs first-generation store-and-forward communication, the effect is ameliorated by means of a virtual channel system of message-passing [39] which simulates second-generation wormhole communication. Packetization reduces large messages, and message aggregation reduces message variance. The communication diameter is also small. The measured communication time was found to be a linear function of message size. In the tests, the farmer was placed on a transputer and each of the eight i860s hosted a worker process, using an incomplete binary tree topology.

Four hundred jobs were sent out on demand for each test. Buffering was turned off and the message size was restricted to a 12 byte tag and 16 bytes data. 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. These results are in a form suitable for the prediction equations. A software synchronized clock with worst-case error of 0.5 ms [40] was employed. The overall time was taken at the central task farmer. A central per-message overhead was introduced by waiting on the low-priority transputer clock, one tick per 0.64 ms, according to a truncated normal distribution. The intention of the overhead was to model both the global processing and/or the reassembly of messages before further processing, an activity often performed at the farmer. An exponential cdf ($1 - e^{-\lambda x}$, $x > 0$) job duration distribution was used, which is not symmetrical but is mathematically tractable. Therefore, the Factoring results do not represent an ideal regime.

In Fig. 3, a task size of six is in most cases a clear minimum. The relative depth of the minimum is governed by the extent of the potential saving. Larger task sizes than six cause the remainder portion, i.e. the time waiting for the last task to complete, to exceed any saving

made by reducing the number of messages. Smaller task sizes than six result in more central overhead. The message-passing overhead remains approximately constant across task size. As to sensitivity to task size, it can be seen from Figure 3, that, within the range of task sizes surveyed according to central overhead, a difference of between about 3 & 10% is possible.

In Fig. 4, a factoring scheduling regime using (16) is compared to the worst and best case task-size regime over a wider range of central workloads. Factoring appears to be a minimal regime though it does not in this instance exceed the performance of uniform task-size scheduling. A simple estimate of the minimum time can be taken from

$$t_{factor} = \frac{\mu n}{p} + vh, \tag{17}$$

where v is the number of scheduling rounds from factoring. In Table 1, the minimal time estimates arising from equations 14 & 17 are compared. The two estimates form upper and lower bounds for the timings.

An exact result for the exponential cdf (here $\lambda = 1$) is available by solving (3)

$$\mu_p = \sum_{i=2}^{p} \frac{1}{i} = \ln p + \gamma + O(p^{-1}), \tag{18}$$

with γ being Euler's constant $(0.5772157\ldots)$[6]. By using (7) with (18) substituted for $E[R_{ms}]$ to yield it is also possible to use a distribution-specific estimate of the optimal task size

$$k_{min_{ms}} = \left(\frac{nh}{p\sigma(\ln(p-1) + \gamma)} \right)^{2/3}. \tag{19}$$

The resulting estimate of k_{opt} in that it rises less steeply appears to be more accurate than $k_{min_{kw}}$, but the estimate of $t_{min_{ms}}$ is too pessimistic. Refer again to Table 1 for the timings.

5.2 Simulation results

A simulation was conducted to verify the large p behaviour of the various estimators. To approach the population statistics 20,000 jobs were distributed. Experiments were made at eight processor intervals from 8 to 248 processors.

[6]Equation 18 will be recognized as a variant of Riemman's zeta function.

Figs. 5–7 are the result, for a sample range of processor numbers, of varying the task size from 1 to 8 jobs with exponential service distribution. In regard to uniform task sizes, for small per-task delays there is a minimum shared between 1 and 7 tasks (not distinguishable on Figs. 5–6). The timings for 2, 3 and 6 tasks are bunched together and cannot be distinguished on the plots. The relative order changes with the number of processors deployed. At the largest delay the order changes to favour sizes 4 and 6 though the results vary considerably with the number of processors. From the simulation it is again apparent that it will be difficult to accurately predict the best task size on an *a priori* basis. Turning to factoring, it is apparent that as the per-task delay increases factoring more clearly represents a minimal regime.

In Fig. 8, the optimal time predictions of Section 5 are tested against the simulation results. For the smallest delay, 0.001, it is possible for the predictors to find a task size of less than one, with the result that the predictions for the optimal time are less than the simulated time. Conversely, for larger delays, here 0.1, the estimates move in the opposite direction.

6 Ongoing work

Given a scheduling scheme that is characterised by second-order statistics and a single communication metric, it becomes feasible to predict favourable scheduling regimes across a range of multicomputers. The PTRAN compiler-based scheduling tool [41] used a built-in set of instruction timings for different machines in order to arrange program runs. Also included with a program were timings, means and variances, found by way of a counter-based profiler. Broadly, a similar scheme can be built into a data-farm template [42]. Where data-farms are arranged in a pipeline [43], if the stages of the pipeline are not synchronous the task scheduling scheme should consider the global flow of tasks across the pipeline. For asynchronous pipeline systems, a promising approach is minimization of a simple model of message-size granularity [44] intended for use within a data-parallel programming paradigm.

On computer systems without a benign communication regime, it may become neces-

sary to employ a hierarchy of data farms, each employing its own scheduling scheme. On NUMA machines, in order to reduce access contention to synchronization variables, a distributed scheduling scheme, affinity scheduling [45], has been suggested. In [46], an hierarchical scheduling scheme is proposed which will scale-up for large systems. Experiments with data-farms on a 24 transputer Parsys supernode showed that performance was improved if tasks were distributed in advance of requests when data farming [47]. Therefore, for large p it may be necessary to modify the raw data-farming system to a hierarchy of data farms.

7 Conclusions

Processor (or task) farming is a commonly used parallel programming paradigm which has the potential for automatic run-time load balancing. A critical parameter in optimising parallel performance is the size of task which is farmed out. A common approach is to use constant-sized tasks defined by the application characteristics, but there is no guarantee that this will produce optimal performance. To obtain optimal design solutions we adapted fine-grained loop-scheduling on NUMA multiprocessors to medium-grained data farming on distributed memory multicomputers. This system of scheduling is appropriate for any application where task durations can be statistically characterised. The paper explores the theory behind order statistics considering firstly a distribution-free estimate of the optimal uniform task size, and secondly a general-purpose decreasing task-size scheduling system, 'Factoring'. It is shown that, as the mean value of the order statistics of common symmetric distributions are approximately triangular in distribution, the Factoring approach, in which the task size decreases exponentially by a factor of two, produces more reliable estimates of optimal performance than the estimators for uniform task size, particularly if the central processing overhead is large. By inclusion of this form of scheduling, the range of applications catered for by the data farm is extended to those with data dependency, for example in higher-level image processing and vision.

Practical timings on a small distributed-memory multicomputer and a simulation across

a wide range of processor numbers were used to test the various estimators. The tests showed that where a decreasing task size is not feasible, nonetheless the factoring prediction may still be used to indicate a minimal regime, even though the job service time distribution under test does not have a symmetric distribution.

The conclusions from this work can be integrated with designs for data farm templates. If the task size can vary across a run Factoring is the preferred choice of task scheduling; if tasks cannot be varied in size, uniform scheduling with task size guided by a prediction of optimal performance by Factoring is chosen.

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] 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 No. 272.

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

[3] 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.

[4] 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.

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

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

[7] 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.

[8] 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.

[9] 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.

[10] D. E. Knuth. *The Art of Computer Programming*, volume 2/ Seminumerical Algorithms. Addison-Wesley, Reading, MA, 2 edition, 1981.

[11] A. G. Chalmers and D. J. Paddon. Parallel radiosity methods. In *Transputer Research and Applications 4*, pages 183–193. IOS, Amsterdam, 1990.

[12] M. Hamdi and C. K. Lee. Dynamic load-balancing of image processing applications on clusters of workstations. *Parallel Computing*, 22(11):1477–1492, January 1997.

[13] D. M. N. Prior, M. G. Norman, N. J. Radcliffe, and L. J. Clarke. What price regularity? *Concurrency: Practice and Experience*, 2(1):55–78, March 1990.

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

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

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

[17] 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.

[18] 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.

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

[20] H. El-Rewini, T. Lewis, and H. H. Ali. *Task Scheduling in Parallel and Distributed Systems*. Prentice-Hall, Englewood-Cliffs, NJ, 1994.

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

[22] 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.

[23] 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.

[24] S. F. Hummel, E. Schonberg, and L. E. Flynn. Factoring: A practical and robust method for scheduling parallel loops. In *Supercomputing Conference*, pages 610–619, November 1991.

[25] 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.

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

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

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

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

[30] 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.

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

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

[33] M. Fleury, A. C. Downton, and A. F. Clark. Performance metrics for pipelined processor farms. *IEEE Transactions on Parallel and Distributed Systems*, 1999. Submitted for publication.

[34] S. F. Hummel, E. Schonberg, and L. E. Flynn. Factoring: A method for scheduling parallel loops. *Communications of the ACM*, 35(8):90–101, 1992.

[35] C. D. Polychronous and D. J. Kuck. Guided self-scheduling: A practical scheduling scheme for parallel supercomputers. *IEEE Transactions on Computers*, 36(12):1425–1439, December 1987.

[36] T. H. Tzen and L. M. Ni. Trapezoid self-scheduling: A practical scheduling scheme for parallel compilers. *IEEE Transactions on Parallel and Distributed Systems*, 4(1):87–98, January 1993.

[37] F. N. David and N. L. Johnson. Statistical treatment of censored data part I fundamental formulæ. *Biometrika*, 41:228–240, 1954.

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

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

[40] 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.

[41] V. Sarkar. Determining average program execution times and their variance. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 298–312, 1989.

[42] M. Fleury, A. C. Downton, and A. F. Clark. Constructing generic data-farm templates. *Concurrency: Practice and Experience*, 11(9):1–20, 1999.

[43] 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.

[44] F. Desprez, P. Ramet, and J. Roman. Optimal grain size computation for pipelined algorithms. *Lecture Notes in Computer Science*, 1123:165–172, 1996. Proceedings of Euro-Par'96 Volume 2.

[45] E. P. Markatos and T. J. LeBlanc. Locality-based scheduling for shared-memory multiprocessors. In A. Y. Zomaya, editor, *Parallel Computing: Paradigms and Applications*, pages 237–276. Thomson, London, 1996.

[46] 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.

[47] M. Fleury and L. Hayat. Performance on a reconfigurable message-passing parallel processor using the data-farming paradigm. Technical report, Essex University, 1993.

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

A An outline derivation of Kruskal-Weiss prediction equation

The approximation to m_p, (11), is derived (in non-trivial fashion) from (8) by setting

$$(1 - H(m_p))^{1/k} = P(s_k \geq m_p)^{1/k} = \left(\frac{1}{p}\right)^{1/k} \approx 1 - \frac{\ln p}{k}, \tag{20}$$

where s_k is the execution time of a task of size k, and H is the cdf of the execution times. (The final approximation in (20) is found by manipulating the Taylor series approximation to $\ln(1-x)$.) For large k

$$\overline{x} = \frac{s_k}{n/p} \approx s_k/k. \tag{21}$$

By the theory of large deviations [48, p. 184],

$$P(s_k \geq k\mu) \approx (\inf_{\theta} G(\mu, \theta))^k, \tag{22}$$

where $G(\mu, \theta) = \int e^{\theta(x-\mu)} dF(x)$, the moment generating function of $X - \mu$. Finally, (11) is arrived at by a Taylor expansion of $G(\mu, 0)$.

B A Factoring regime derivation

At round i, assuming $p \gg 1$,

$$r_{i+1} = r_i - pk_i,\ k_i = r_i/(yp), \tag{23}$$

where r_i is the total number of tasks remaining at each scheduling round, $r_0 = n$ and y is unknown. To achieve a balance requires, from (15),

$$\mu k_{i+1} = \sigma\sqrt{k_i p/2}. \tag{24}$$

If

$$\begin{aligned} r_{i+1} &= r_i(1 - 1/y) \\ k_{i+1} &= \tfrac{r_i(1-1/y)}{yp}, \end{aligned} \tag{25}$$

are substituted in (24), after rearrangement,

$$\frac{2(1-1/y)^2}{y} = \left(\frac{\sigma}{\mu}\right)^2 \frac{p^2}{r_i}. \tag{26}$$

Notice that (26) depends on the c.o.v. The L.H.S. of (26) is at a minimum at $y = 1$, and can be approximated by a Taylor expansion close to one. A conservative estimate is

$$y \approx 1 + 2\left(\frac{\sigma}{\mu}\right)^2 \frac{p^2}{r_i}. \tag{27}$$

With $x = 2$ one has the Factoring regime.

Central workload index	Predicted task size		Actual task size	Actual min. time	Uniform estimate		Factoring estimate
	$k_{opt_{kw}}$	$k_{opt_{ms}}$	k_{actual}		$t_{min_{kw}}$	$t_{min_{ms}}$	$t_{min_{fact}}$
1	2.1	1.5	2	0.8322	0.8489	0.8599	0.8004
5	2.9	2.0	6	0.8377	0.8570	0.8697	0.8061
10	3.7	2.6	6	0.8406	0.8647	0.8791	0.8093
20	5.1	3.5	6	0.8500	0.8762	0.8931	0.8157
30	6.3	4.4	6	0.8619	0.8850	0.9039	0.8221
40	7.4	5.2	6	0.8712	0.8923	0.9128	0.8285
50	8.5	5.9	6	0.8811	0.8986	0.9205	0.8349

Table 1: Predictions for minimum job duration (s)

List of Figures

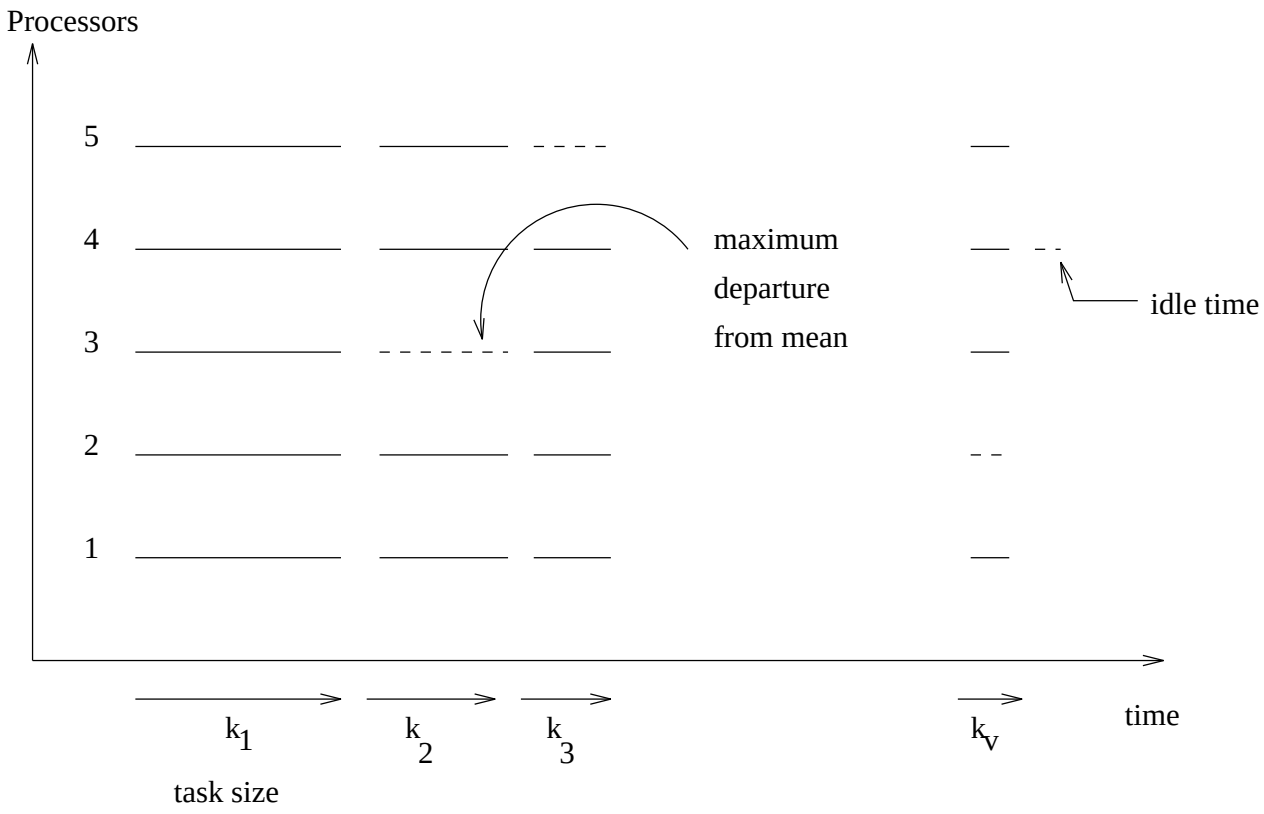

Figure 1: Decreasing task-size method of scheduling

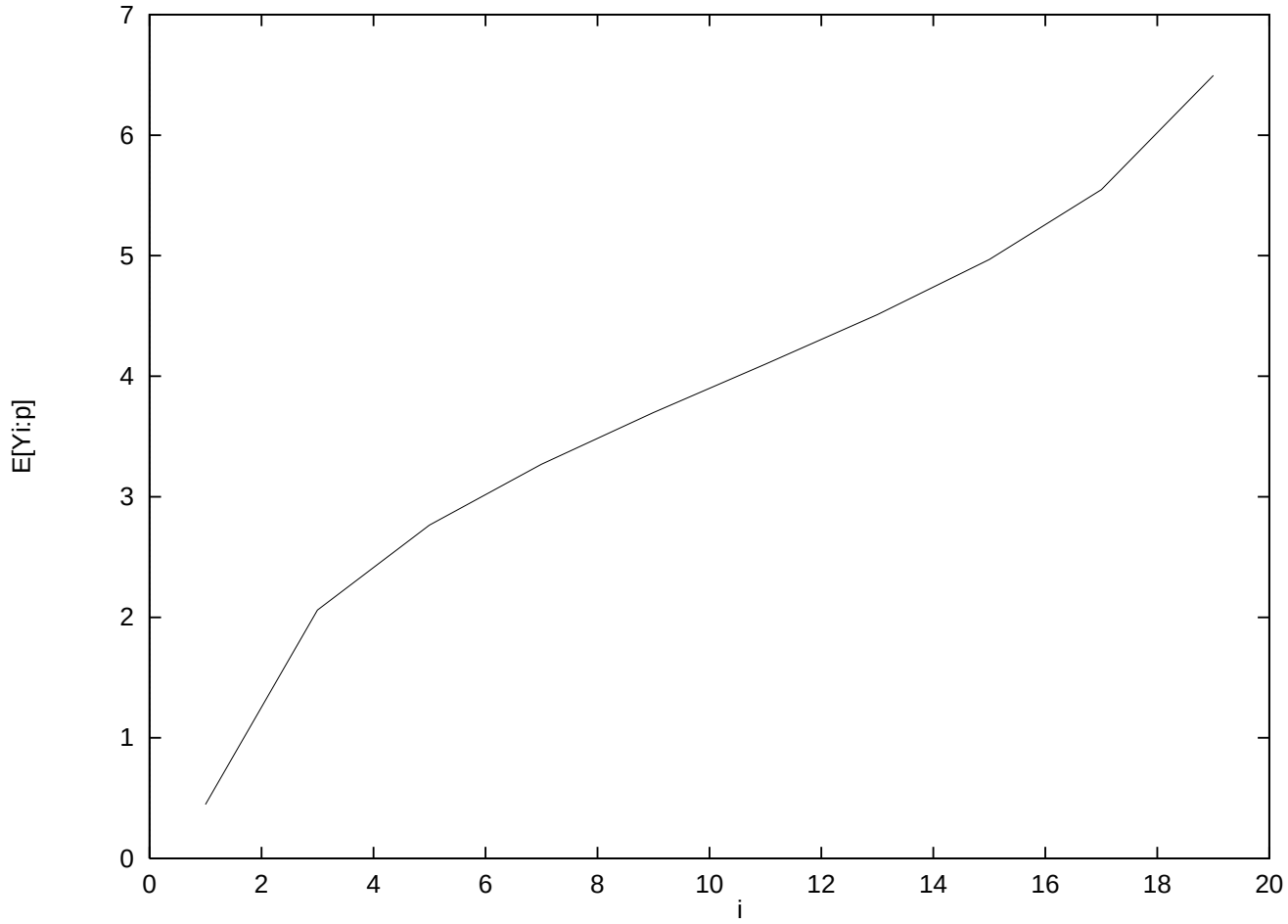

Figure 2: Approximate mean order statistics for the Logistic distribution

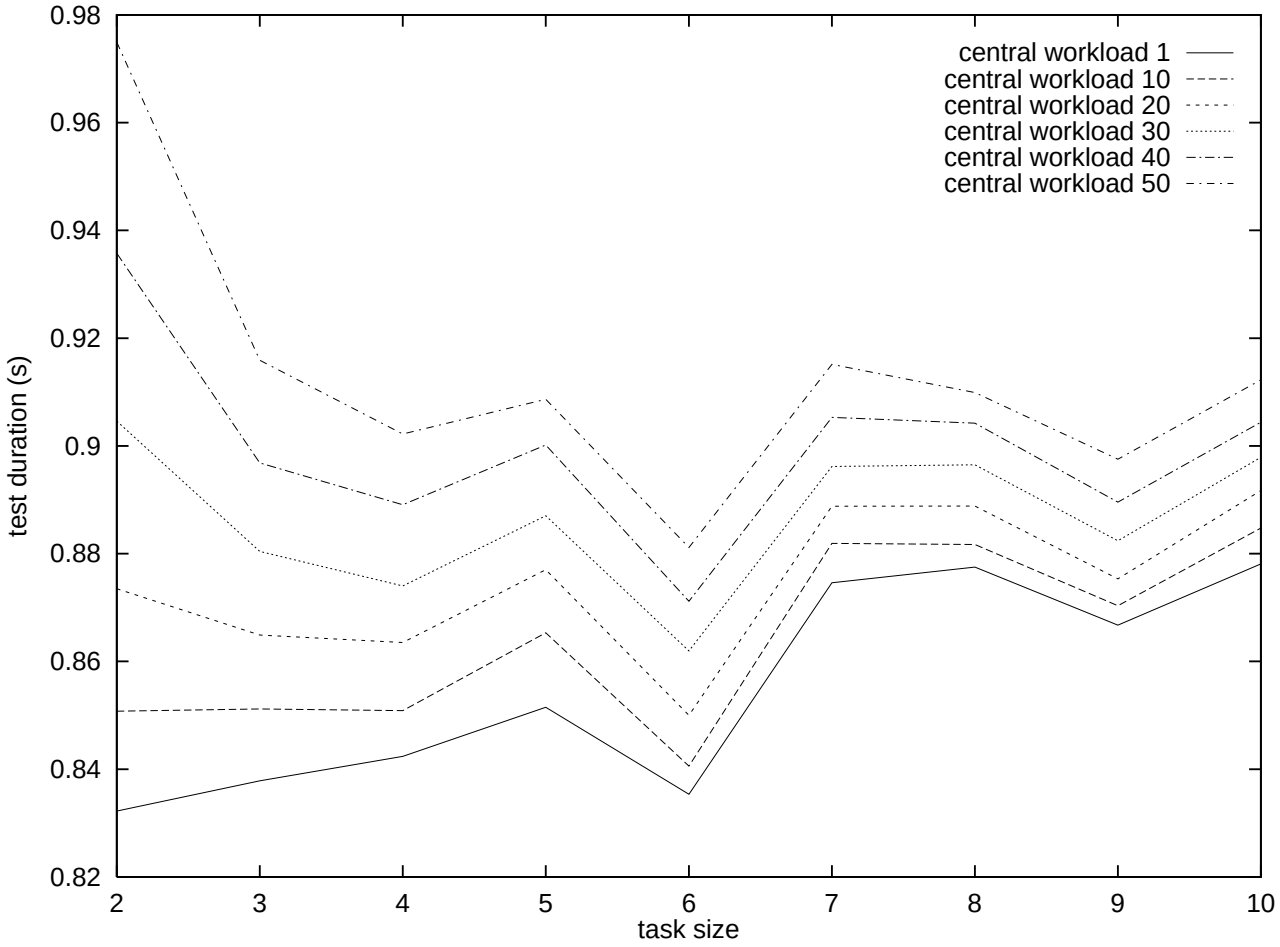

Figure 3: Uniform task-size scheduling

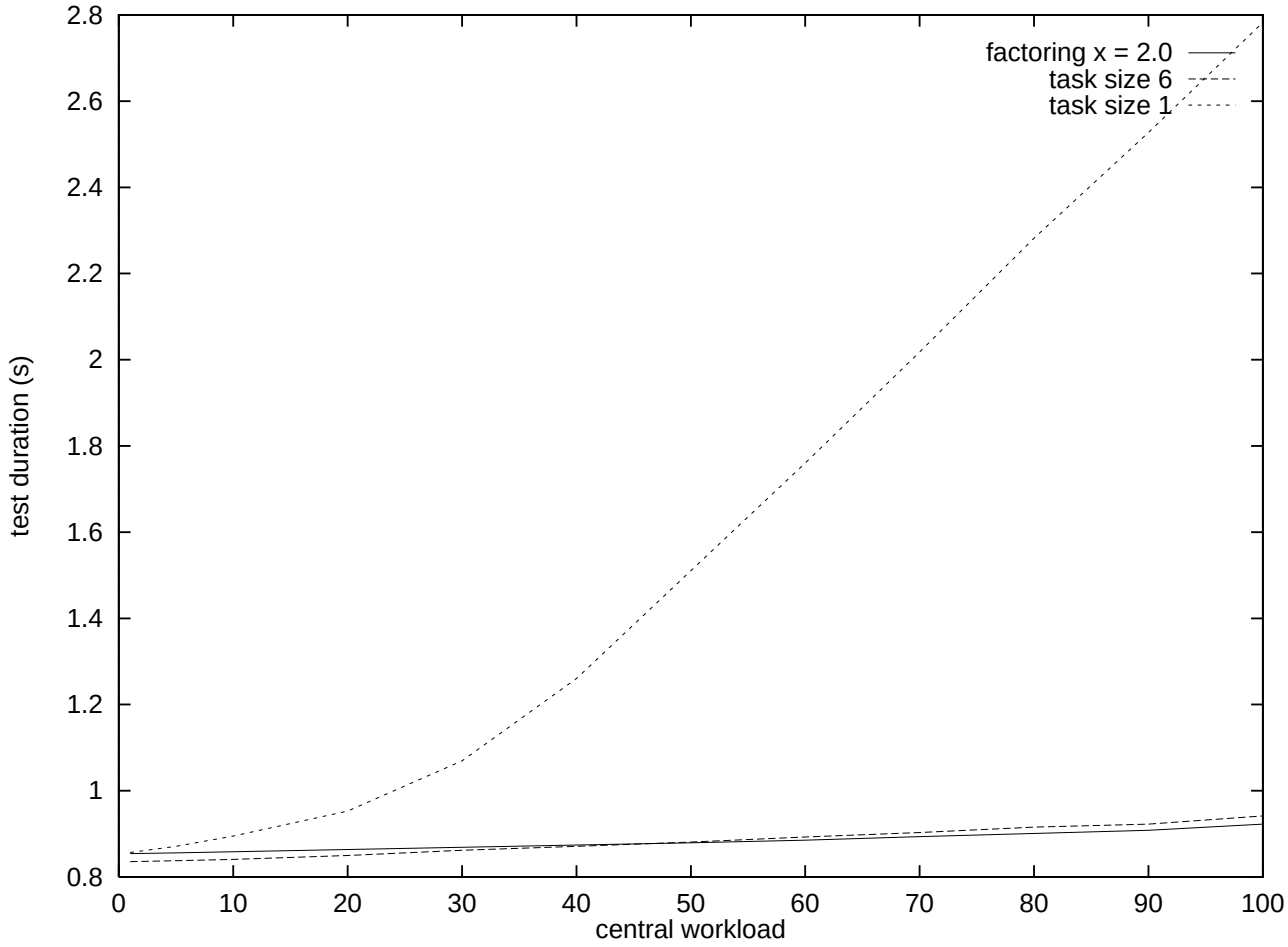

Figure 4: Comparison of scheduling regimes

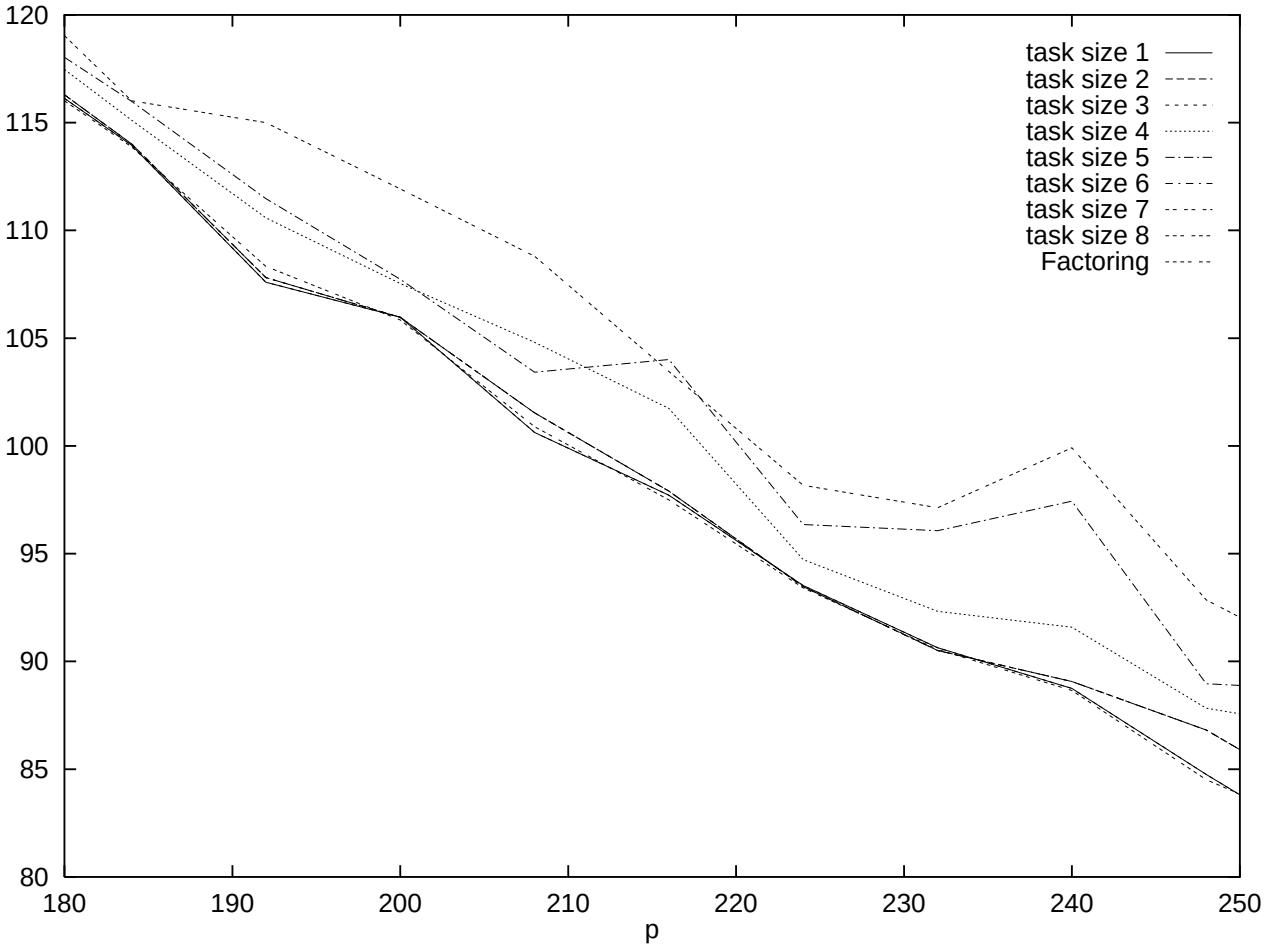

Figure 5: Simulation of uniform task-size scheduling, delay 0.001

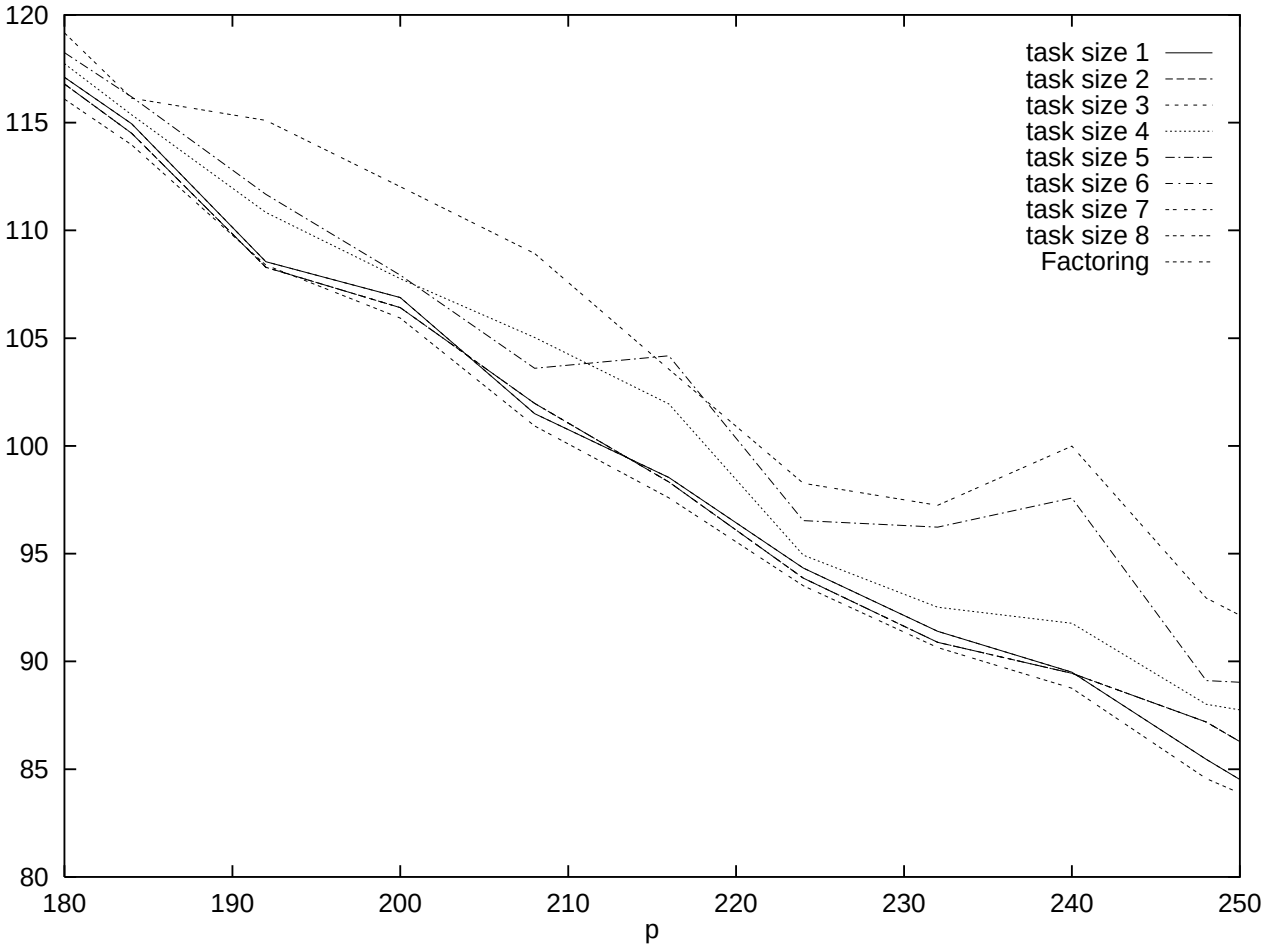

Figure 6: Simulation of uniform task-size scheduling, delay 0.01

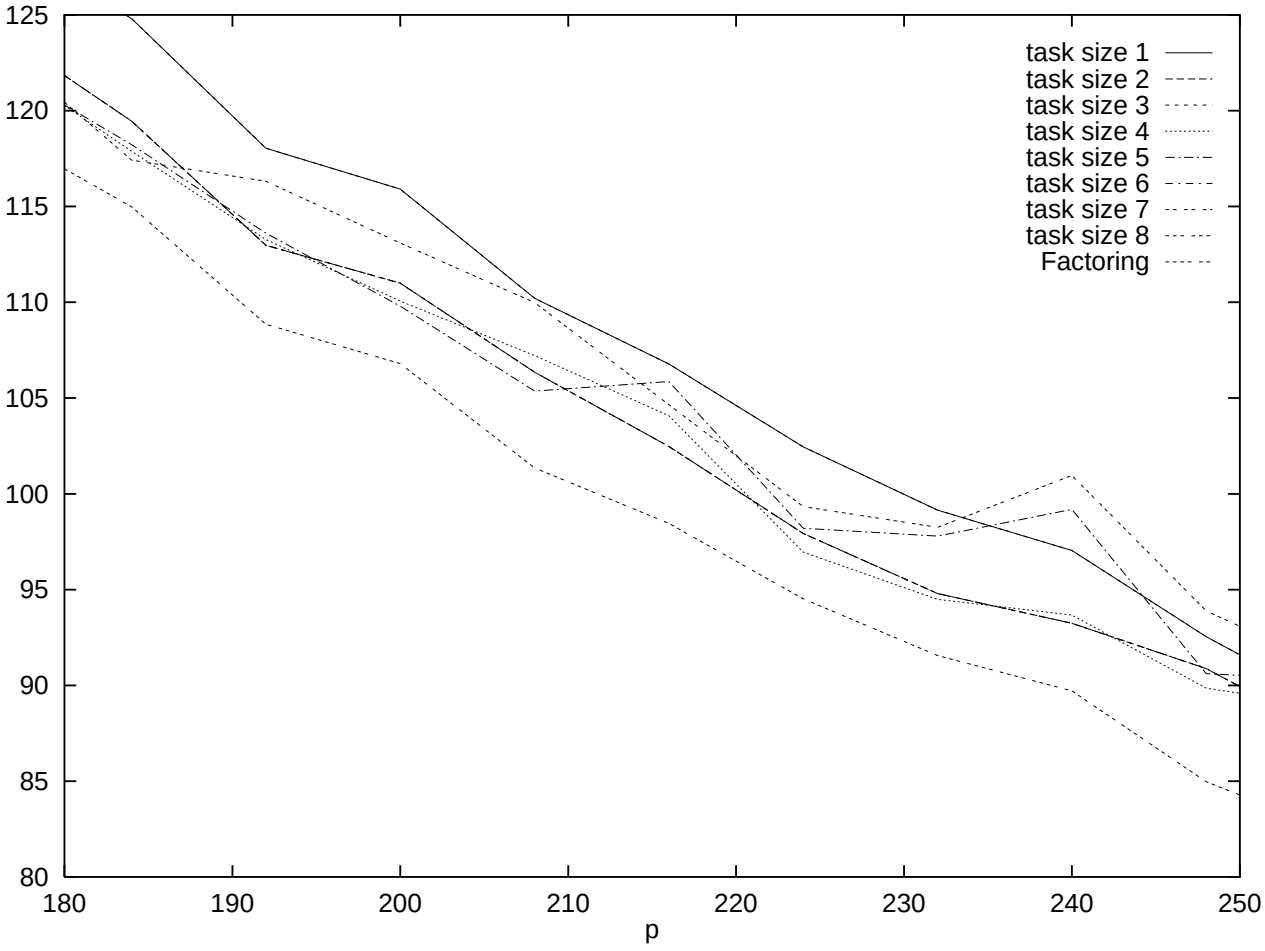

Figure 7: Simulation of uniform tasks, delay 0.1

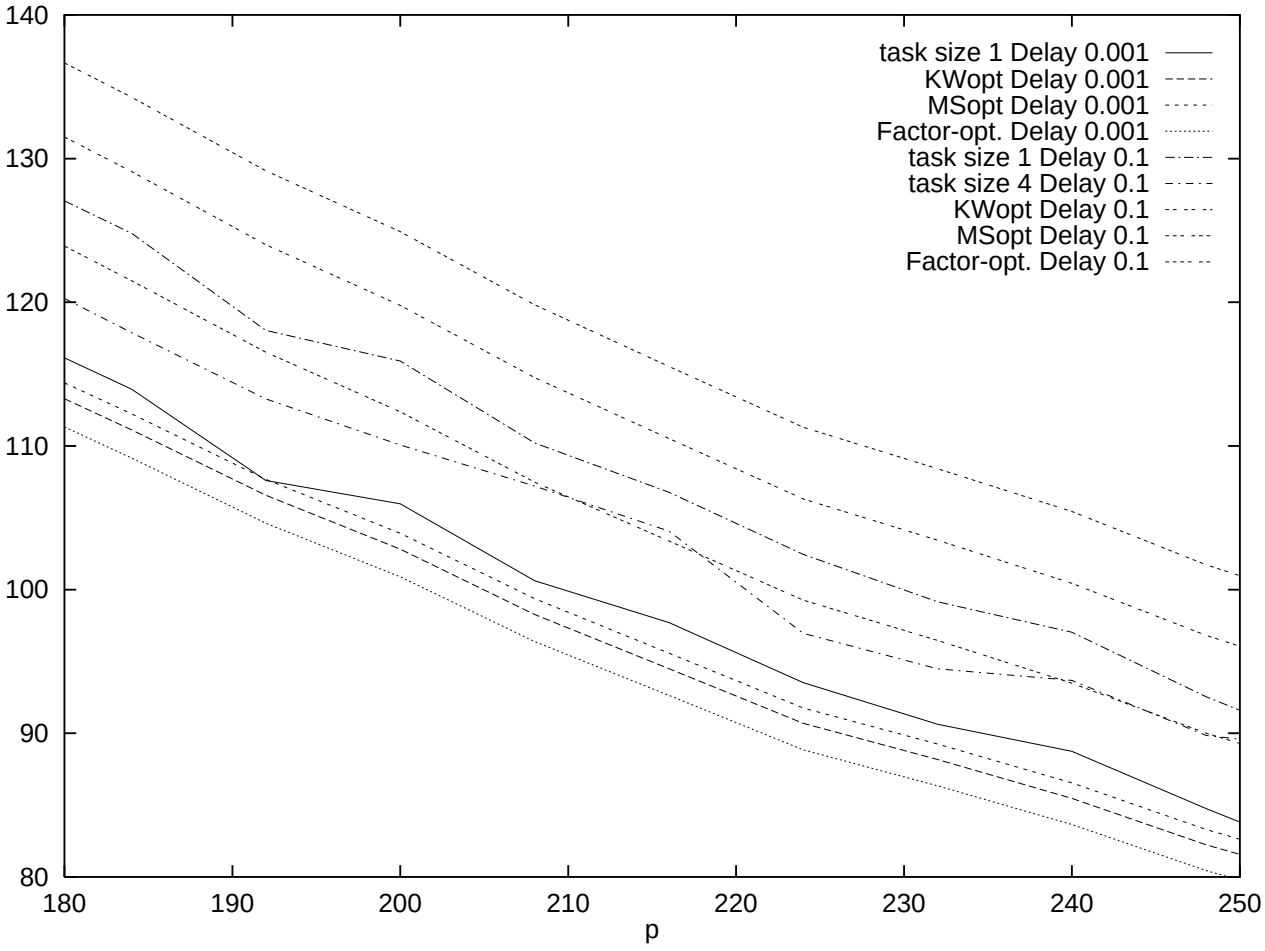

Figure 8: Predictions of optimal times