In: Massively Parallel Processing Applications and Development (L. Dekker, W. Smit and J.C. Zuidervaart, eds.), North-Holland, 1994, pp. 389–396.

Efficient performance evaluation of parallel systems

H. Jonkers, A.J.C. van Gemund, G.L. Reijns

Delft University of Technology, Faculty of Electrical Engineering
P.O. Box 5031, 2600 GA Delft, The Netherlands
{h.jonkers, a.vgemund, g.l.reijns}@et.tudelft.nl

For the effective development of applications running on massively parallel systems, efficient but reliable performance predictions are required. However, existing performance evaluation formalisms are either not directly suitable for modelling parallel applications, or are too computationally expensive. To alleviate these problems, two novel methodologies have recently been introduced, one based on fast deterministic predictions and one based on traditional probabilistic performance modelling (queueing theory). This paper for the first time compares these two methodologies to each other, based on a distributed-memory case study. Both turn out to have their own merits, and depending on the purpose of the performance predictions the most appropriate methodology can be chosen.

Keywords: Performance evaluation, distributed-memory computing, case study, resource contention, serialization analysis, closed queueing networks.

1. INTRODUCTION

Given the need to integrate performance analysis at various stages of the parallel application design process, a performance modelling methodology which combines low cost with a high-reliability prediction becomes a critical factor in exploiting the potential of high-performance computing. In performance modelling, a trade-off exists between prediction reliability and efficiency of analysis, resulting in a performance prediction hierarchy. In case of parallel processing, traditional performance modelling techniques are either unreliable or entail prohibitive evaluation cost. Although many extensions have been proposed and a large number of application-specific models have been developed, no unified

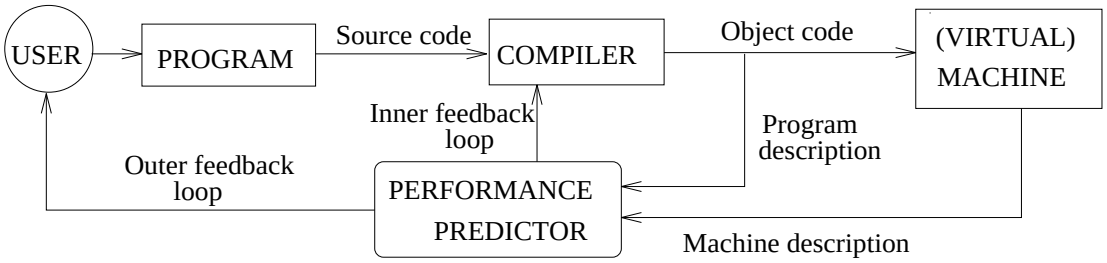

Figure 1. Parallel programming environment with performance feedback

methodologies have yet emerged. In order to alleviate these problems two new modelling methodologies have been introduced separately, PAMELA and *Glamis*, each representing a different position on the trade-off. Given a parallel application development environment with performance feedback, as shown schematically in figure 1, PAMELA is most eligible for the inner feedback loop (compile-time optimization), while *Glamis* is more appropriate for the outer feedback loop.

Within the PAMELA methodology [2] the emphasis is on static prediction which inherently yields fast but relatively inaccurate performance models. While current compile-time estimation techniques do not account for contention, PAMELA features an approximative contention analysis method (serialization analysis) which introduces a fundamentally higher prediction reliability without cost penalty. Due to the static approach, PAMELA models are typically symbolic, thus allowing for a full parameter study without recompilation.

Glamis [4], on the other hand, extends probabilistic performance modelling techniques (in particular queueing networks), yielding more accurate (although often numerical instead of closed-form) predictions at the expense of somewhat higher analysis costs, which however are still polynomial as opposed to some alternative probabilistic techniques. Contrary to standard queueing models, *Glamis* allows for the specification and analysis of certain kinds of condition synchronization.

Both PAMELA and *Glamis* exploit inherent *replications* in parallel systems to further reduce the analytical complexity of the models. An additional advantage of this approach is that it results in models that are *scalable*. This is of particular importance for massively parallel systems, which generally exhibit a very high degree of replication. While the principles of both methodologies have been published separately, and case studies of shared-memory applications have been reported, this paper compares the results of the two methodologies to each other, based on a distributed-memory (MPP) case study. Because of space limits, we refer to the previous publications for an overview of PAMELA and *Glamis*.

The remainder of this paper is organized as follows. In section 2, the case study is introduced. Section 3 presents the PAMELA model and serialization analysis, while in section 4 the *Glamis* model and its analysis are treated. Section 5 present the results of the different prediction methods and compares them to simulations. These results are discussed in section 6. Finally, in section 7 some conclusions are drawn.

2. CASE STUDY: MATRIX-VECTOR UPDATE

Consider a matrix-vector update $y \leftarrow y + Ax$, carried out on a distributed-memory architecture with P processing nodes interconnected in a unidirectional point-to-point ring topology. The architecture (for $P = 6$) is shown in figure 2.

The $N \times N$ matrix A is block-partitioned over the processors, in an intentionally suboptimal column-wise way in order to emphasize the role of interprocessor communication. Every processor owns $b = N/P$ columns (b is the block size), assuming $P|N$. The SPMD-(pseudo)code is listed in figure 3. In this code, SEND(p', u) (synchronously) moves a data element u from node p to node p', while RECV(p') returns a data element sent by node p' to node p.

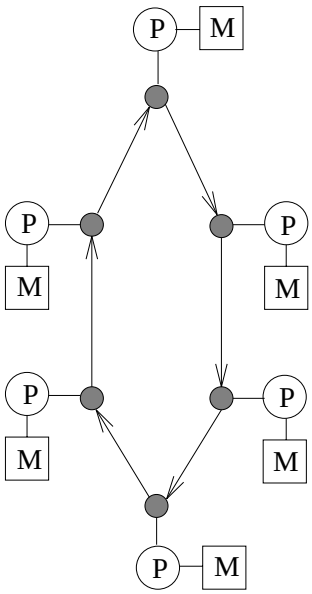

Figure 2. Ring architecture

```
node(p):
myl = b * p; myu = b * (p + 1) − 1;
for i = 0 ... myl − 1
do for j = myl ... myu
   do SEND(i/b, A[i, j]);
for i = myu + 1 ... N − 1
do for j = myl ... myu
   do SEND(i/b, A[i, j]);
for i = myl ... myu
do {
     for j = myl ... myu
     do y[i] := y[i] + A[i, j] * x[j];
     for j = 0 ... myl − 1
     do y[i] := y[i]+RECV(j/b)*x[j];
     for j = myu + 1 ... N − 1
     do y[i] := y[i]+RECV(j/b)*x[j];
   }
```

Figure 3. SPMD code of matrix-vector update

In the example, we use the following parameter settings: the duration of one floating-point operation is $t_f = 0.200$ time units, the access time of a link is $t_l = 0.025$ time units, $N = 60$, and P ranges from 1 to 60 with $P|N$.

3. PAMELA MODEL AND SERIALIZATION ANALYSIS

Corresponding to the procedural modelling paradigm, condition synchronizations are ignored in the PAMELA model. Furthermore, we only model the SEND statements and floating-point operations, assuming that the RECV operation only accounts for synchronization and local data transfers. The resulting PAMELA model is

$$
\begin{aligned}
L = \textbf{par}\ (p = 0 \ldots P-1)\ \{ & \\
& \textbf{seq}\ (i = 0 \ldots bp-1)\ \textbf{seq}\ (j = bp \ldots b(p+1)-1)\ \mathit{send}(i/b); \\
& \textbf{seq}\ (i = b(p+1) \ldots N-1)\ \textbf{seq}\ (j = bp \ldots b(p+1)-1)\ \mathit{send}(i/b); \\
& \textbf{seq}\ (i = bp \ldots b(p+1)-1)\ \{ \\
& \quad \textbf{seq}\ (j = bp \ldots b(p+1)-1)\ \{\ \mathit{flop};\ \mathit{flop};\ \} \\
& \quad \textbf{seq}\ (j = 0 \ldots bp-1)\ \{\ \mathit{flop};\ \mathit{flop};\ \} \\
& \quad \textbf{seq}\ (j = b(p+1) \ldots N-1)\ \{\ \mathit{flop};\ \mathit{flop};\ \} \\
& \} \\
\}
\end{aligned}
$$

The machine model associated with the ring architecture is simply:

$$\begin{aligned} \mathit{flop} &= \mathbf{delay}(t_f) \\ \mathit{send}(p') &= \mathbf{seq}\ (k = 0 \ldots K-1)\ \mathbf{use}(l_{(p+k) \bmod P}, t_l) \end{aligned}$$

where $K = (P + p' - p) \bmod P$ denotes the number of links involved in the transmission (forwarding costs are ignored). Thus, the links are assumed to be the only sources of contention.

According to the serialization transformation (transformation 2) introduced in [1], the PAMELA prediction for the completion time T_{pam} equals

$$T_{\text{pam}} = \max\{T_{\text{cplx}}, \sigma_{\max}\}$$

where T_{cplx} is the completion time obtained with complexity analysis (ignoring link contentions) and $\sigma_{\max}$ is the maximum workload on a link. Complexity analysis makes use of the following model of *send*

$$\mathit{send}(p') = \mathbf{seq}\ (k = 0 \ldots K-1)\ \mathbf{delay}(t_l) = \mathbf{delay}(Kt_l)$$

with $K = (P + \lfloor i/b \rfloor - p) \bmod P$. Traditional analysis yields

$$T_{\text{cplx}} = \max_{p=0}^{P-1} \left(\sum_{i=0}^{bp-1} \sum_{j=0}^{b-1} Kt_l + \sum_{i=b(p+1)}^{N-1} \sum_{j=0}^{b-1} Kt_l \right) + 2bNt_f$$

which eventually reduces to

$$T_{\text{cplx}} = b^2 \frac{P(P-1)}{2} t_l + 2bNt_f$$

In order to derive the link workloads needed for serialization analysis, we consider the visit count V_l on links $l_0 \ldots l_{P-1}$. From the structure of L and *send* it immediately follows

$$V_l = \sum_{p=0}^{P-1} \left(\sum_{i=0}^{bp-1} \sum_{j=0}^{b-1} \sum_{k=0}^{K-1} c + \sum_{i=b(p+1)}^{N-1} \sum_{j=0}^{b-1} \sum_{k=0}^{K-1} c \right)$$

where

$$c = [(p+k) \bmod P = l] \quad \text{and} \quad K = (P + \lfloor i/b \rfloor - p) \bmod P$$

This formula eventually reduces to

$$V_l = b^2 \frac{P(P-1)}{2}$$

Thus, for each link the workload $\sigma_{\max}$ is equal and given by

$$\sigma_{\max} = b^2 \frac{P(P-1)}{2} t_l$$

By the above transformation it holds

$$T_{\text{pam}} = \max \left\{ b^2 \frac{P(P-1)}{2} t_l, b^2 \frac{P(P-1)}{2} t_l + 2bNt_f \right\}$$

which implies that serialization yields the following execution time, which is equal to the result of complexity analysis:

$$T_{\text{pam}} = T_{\text{cplx}} = N^2 \left(\frac{P-1}{2P} t_l + \frac{2}{P} t_f \right)$$

For a more complete derivation of these expressions we refer to [1].

4. *Glamis* MODEL AND ANALYSIS

Again, we will assume that the links are the only sources of contention. Therefore, every link is modelled by a single queueing centre. Other work is local to a processor, and is modelled by a delay centre. A regular queueing model is shown in figure 4, where the circles labelled D represent a delay centre.

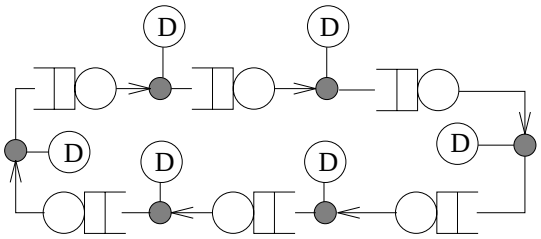

Figure 4. Queueing model ($P = 6$)

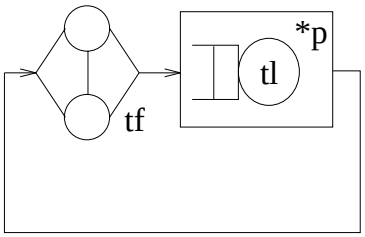

Figure 5. *Glamis* machine model

In the *Glamis* model, the symmetries present in the queueing model are exploited in order to reduce the complexity. All the delay centres can be merged into one infinite server centre. Although for every processor separately the workload on all of the links is different (the link nearest to the processor, in the transmission direction, is most frequently accessed), all links experience the same total workload. In order to keep the model tractable, an even distribution is assumed. Because the average workload per processor is the same for every link, this assumption does not significantly violate the real situation (especially when a probabilistic, exponentially distributed, service time is used, which more or less accounts for the variation in the service demands). The number of links is equal to the number of processors, so that the queues representing the links can be combined into one block of equivalent queueing centres. The optimized *Glamis* queueing model is shown in figure 5, or in tuple notation $\langle I(t_f), Q^p(t_l)\rangle$. The time needed to receive a message is assumed to be incorporated in the floating-point times, and is not explicitly modelled.

The program consists of one parallel section of P tasks, every task executing the SPMD-program once. Every task performs $2N^2/P$ floating-point operations and $(N/P)^2(P-1)$ SENDs. The number of link accesses ("hops") during a SEND ranges from 1 to $p-1$, every number of hops occurring with the same frequency. This results in an average link visit count of $\frac{1}{2}(N/P)^2(P-1)$. The resulting program model is

$$P : \left\langle 2N^2/P, (N/P)^2(P-1)/2 \right\rangle$$

In this case, the assumption made in MVA that the order of the resource usages is irrelevant will lead to an under-estimation of the contention delay. This is due to the fact that the SENDs are completely concentrated in the beginning of the tasks and the floating-point operations in the end of the tasks. Especially for a small number of processors this effect is notable. Because of the total separation of the SENDs and the floating-point operations, the tasks can be split in two, as shown in figure 6 (the shaded areas denote

communications, the open areas computations). The resulting graph does not have an SPS-structure (*sequence of parallel sections* [4]), but its structure is approximately SPS, so that approximate results can be obtained by inserting a barrier synchronization. A trace of the simulation results shows that there is hardly any overlap of communications and computations. Therefore, the approximations are expected to be good. In tuple notation

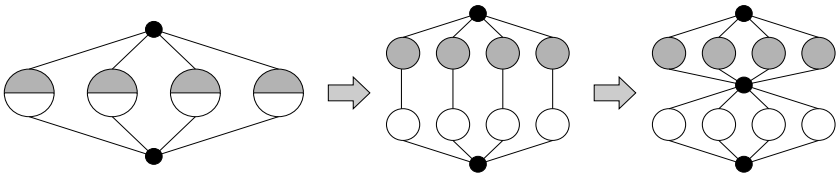

Figure 6. Insertion of barrier synchronization in the task graph

the "split" program model becomes

$$P : \left\langle 2N^2/P, 0 \right\rangle ; P : \left\langle 0, (N/P)^2(P-1)/2 \right\rangle$$

The models can be analysed in a straightforward manner using regular single-class MVA, both for the original program model and the program model with split tasks. In the latter case the exact solution can be expressed in closed-form because of the total separation of floating-point operations and link accesses (SENDs). The duration of the $2N^2/P$ floating-point operations is simply $(2N^2)/Pt_f$, or with the given parameter settings $7200/P$ time units. The total throughput of the block of queues representing the links ($Q^P(t_l)$) with P jobs is $X(P) = P/((2P-1)t_l)$ [4], which results in a response time (i.e. duration of one link access) of $R(P) = P/X(P) = (2P-1)t_l$, which with a total of $N^2(P-1)/(2P^2)$ link accesses gives a completion time of $N^2(P-1)(2P-1)t_l/(2P^2)$, or with the given parameter settings $360(P-1)(2P-1)/P^2$ time units. This yields a total program completion time of

$$T_{\text{split}}(P) = \frac{7200}{P} + \frac{360(P-1)(2P-1)}{P^2} = \frac{360(2P^2+17P+1)}{P^2}$$

In the original situation (without splitting the tasks) a closed-form expression exists for the Bard/Schweitzer approximate solution. Given the general equation for a delay centre and m identical queueing centres derived in [3], and the chosen parameter settings, this results in a completion time of

$$T_{\text{orig}}(P) = \frac{180(2P^2+17P+1+\sqrt{4P^4-12P^3+453P^2-46P+1})}{P^2}$$

5. RESULTS

The results of our models in terms of speedup (i.e. T(1)/T(P)), are shown in figure 7. The predictions are compared to simulation results, which may be considered to be the real ("measured") values, given our aim to evaluate *analysis* techniques rather than actual *modelling* accuracy. The speedups are lower than those that are usually achieved with a

matrix-vector update on an actual ring architecture, due to the intentional sub-optimal data partitioning. In the simulations, all the delays are deterministic. The simulation program has been written in C, making use of the VOP concurrent simulation library [5]. Also the predictions obtained with (parallel) complexity analysis (disregarding link contentions, but including link delays) are included in the picture.

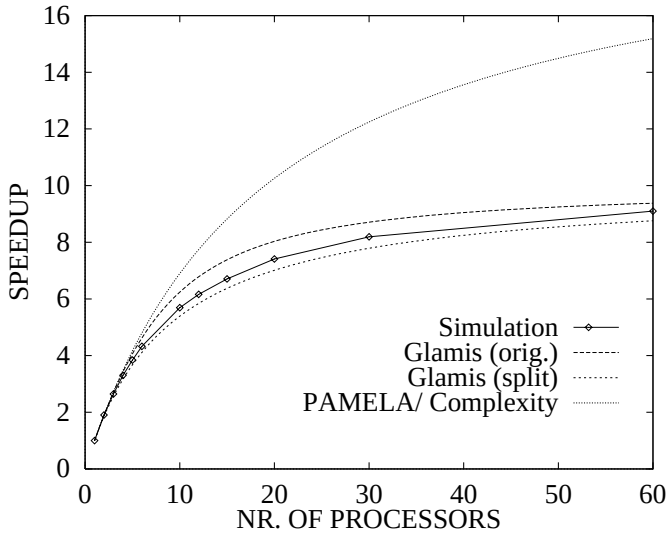

Figure 7. Speedup of matrix-vector update

6. DISCUSSION

Both *Glamis* models are able to predict the speedup well. However, the results of the modified model (with split tasks and barrier synchronization) are slightly more accurate. The original model under-estimates the completion times (and hence over-estimates the speedup values, because the prediction for $P = 1$ is exact), while the modified model over-estimates the completion times. For increasing P both predictions converge to the simulation value, in contrast to complexity analysis. This is due to the fact that the influence of link contentions grows as P increases.

As mentioned in the section 3, in this particular example the PAMELA predictions coincide with those of complexity analysis, which indicates that the effects of contention do not dominate the performance. This is due to the number of (link) resources scaling linearly with the number of processors, thus maintaining balance (note that this is also true for many other distributed-memory architectures, e.g. meshes). Yet, the predictions still considerably over-estimate the simulated speedup. However, the analysis technique can only partly be accounted for this, because scheduling non-determinism can have a large impact on the effects of contention. In this case it even turns out that the communication schedule of the algorithm can be modified in such a way that a completely conflict-free execution is obtained (in which case the prediction is correct). Thus, PAMELA predictions provide diagnostic information indicating the feasibility to optimize the implementation. However, the fact that PAMELA predicts a better performance than that is actually measured, does not in all cases guarantee that a better implementation is possible.

7. CONCLUSIONS

This paper presents a first comparison between the two parallel application performance modelling methodologies PAMELA and *Glamis*, based on a simple distributed-memory case study. Although one example does not allow for a thorough comparison, it does show the main advantages and drawbacks of the two methodologies with respect to each other and to related methodologies.

Both methodologies provide a theoretical framework for modelling and analysing the performance of parallel and distributed systems. When contention dominates, PAMELA's serialization analysis, by accounting for the effects of contention, outperforms traditional static compile-time reduction approaches, at a negligible increase of cost At the expense of a somewhat higher (but still polynomial) analysis cost, *Glamis* enables the analysis of additional dynamic (non-deterministic) aspects, and offers the possibility to *generalize* over classes of machines and programs. While PAMELA always gives closed-form predictions, analysis in *Glamis* is generally numerical (although in the case study presented in this paper also *Glamis* gives closed-form expressions).

Opposed to some other performance modelling methods frequently applied to predict the performance of parallel systems (e.g. Timed Petri Nets), the key target of PAMELA and *Glamis* is efficiency of analysis, which is of crucial importance for the performance evaluation of massively parallel systems. However, their modelling power is still kept sufficiently high to obtain reliable predictions. The way in which our methodologies treat replications (both at the machine level and the program level) results in models that are easily scalable. Another important aspect is the reusability of machine models and program models, which is achieved through a clean separation of the two (in PAMELA machine influences are incorporated in a program model by substitution, while in *Glamis* the separation is accomplished by the definition of a mapping from instructions to visit counts). In the case study presented in this paper, the *Glamis* predictions better approach the actual performance, while PAMELA predicts that it might be possible to construct a better (even totally contention-free) implementation.

REFERENCES

1. A.J.C. van Gemund, "On the analysis of PAMELA models," Tech. Rep. 1-68340-44(1993)05, Delft University of Technology, Delft, The Netherlands, Dec. 1993.
2. A.J.C. van Gemund, "Performance prediction of parallel processing systems: The PAMELA methodology," in *Proc. 7th ACM Int. Conf. on Supercomputing*, Tokyo, Japan, July 1993, pp. 318–327.
3. H. Jonkers, "Queueing models of shared-memory parallel applications," in *Proc. UK Performance Engineering Workshop for Computer and Telecommunication Systems*, Loughborough, U.K., July 1993.
4. H. Jonkers, "Queueing models of parallel applications: The *Glamis* methodology," to appear in *Proc. 7th Int. Conf. on Modelling Techniques and Tools for Comp. Perf. Eval.*, Vienna, Austria, May 1994.
5. R. Pulleman, "Simulation of VOP models," Tech. Rep. 92 TPD-ZP 938, TNO Institute for Applied Physics, Delft, The Netherlands, Sept. 1992.