Optimal Mapping of Pipeline Algorithms[1]

Daniel González, Francisco Almeida, Luz Marina Moreno, Casiano Rodríguez

Dpto. Estadística, I. O. y Computacion, Universidad de La Laguna, La Laguna, Spain
{dgonmor, falmeida, casiano}@ull.es

Abstract. The optimal assignment of computations to processors is a crucial factor determining the effectiveness of a parallel algorithm. We analyze the problem of finding the optimal mapping of a pipeline algorithm on a ring of processors. There are too many variables to consider, the number of virtual processes to be simulated by a physical processor and the size of the packets to be communicated. We provide an analytical model for an optimal approach to these elements. The low errors observed and the simplicity of our proposal makes this mechanism suitable for its introduction in a parallel tool that compute the parameters automatically at running time.

1 Introduction

The implementation of pipeline algorithms on a target architecture is strongly conditioned by the actual assignment of virtual processes to the physical processors, their simulation, the granularity of the architecture, and the instance of the problem to be executed. To preserve the optimality of a pipeline algorithm, a proper combination of these factors must be considered.

The amount of theoretical works [1], [4] contrast with the absence of software tools to solve the problem, most of these solve the case under particular assumptions. Unfortunately, the inclusion of the former methodologies in a software tool is far of being a feasible task.

The *llp* tool presented in [2] allows cyclic and block-cyclic mapping of pipeline algorithms according to the user specifications. We have extended it with a buffering functionality and it is also an objective of this paper to supply a mechanism that allows *llp* to generate automatically the optimal mapping.

[1] The work described in this paper has been partially supported by the Canary Government Research Project PI1999/122.

2 The Problem

The pipeline mapping problem is defined as finding the optimal assignment of a virtual pipeline to the actual processors to minimize the execution time. We consider that the code executed by every virtual process of the pipeline is the standard loop of figure 1. The code of figure 1 represents a wide range of situations as is the case of many parallel Dynamic Programming algorithms [2], [3].

The classical technique consist of partitioning the set of processes following a mixed block-cyclic mapping depending on the Grain G of processes assigned to each processor. According to the granularity of the architecture and the grain size G of the computation, it is convenient to buffer the data communicated into the sender processor before an output is produced. The use of a data buffer of size B reduces the overhead in communications but can introduce delays between processors increasing the startup of the pipeline.

```
void f() {
  Compute(body0);
  While (running) {
    Receive();
    Compute(body1);
    Send();
    Compute(body2);
  }
}
```

Fig. 1. Standard loop on a pipeline algorithm.

We can now formulate the problem: *Which are the optimal values for G and B?*

3 The Analytical Model

Given a parallel machine, we aim to find an analytical model to obtain the optimal values of G and B for an instance of a problem. The time that elapses from the moment that a parallel computation starts to the moment that the last processor finishes executions has to be modeled. This problem has been previously formulated by [2] using tiling. The size of the tiles must be determined assumed the shape. However, the approach taken assumes that the computational *bodies 0* and *2* in the loop are empty. Also, the considerations about the simulation of the virtual processes are omitted.

When modeling interprocessor communications, it is necessary to differentiate between external communication (involving physical processors) and internal communications (involving virtual processors). For the external communications, we use the standard communication model. At the machine level, the time to transfer B words between two processors is given by $\beta + \tau B$, where β is the message startup time and τ represents the per-word transfer time. With the internal communications we assume that per-word transfer time is zero and we have to deal only with the time to access the data. We differentiate between an external reception (β^E) without context switch between processes and an internal communication (β^I) where the context switch must be considered. We will also denote by t_0, t_1, t_{2i} the time to compute respectively *body0*, *body1* and *body2* at iteration *i.*

T_s will denote the startup time between two processors. T_s includes the time needed to produce and communicate a packet of size B. T_c denotes the whole evaluation of G processes, including the time to send M/B packets of size B:

$$T_s = t_0*(G - 1) + t_1 * G * B + G*\Sigma_{i = 1, (B-1)} t_{2i} + 2*\beta^I * (G - 1)* B + \beta^E * B + \beta + \tau *B$$

$$T_c = t_0*(G - 1) + t_1*G*M + G*\Sigma_{i = 1, M} t_{2i} + 2*\beta^I *(G - 1)*M + \beta^E*M + (\beta + \tau*B)* M/B$$

The first three terms accumulate the time of computation, the fourth term is the time of context switch between processes and the last terms include the time to communicate packets of size B.

According to the parameters G and B two situations may appear when executing a pipeline algorithm. After a processor finishes the work in one band it goes to compute the next band. At this point, data from the former processor may be available or not. If data are not available, the processor spends idle time waiting for data. This situation arises when the startup time of processor p (the first processor of the ring in the second band) is larger than the time to evaluate G virtual processors, i. e, when $T_s * p \geq T_c$. Then we denote by R_1 the values (G, B) where $T_s * p \leq T_c$ and R_2 the values (G, B) such that $T_s * p \geq T_c$.

For a problem with N stages on the pipeline (N virtual processors) and a loop of size M (M iterations on the loop), if $1 \leq G \leq N/p$ and $1 \leq B \leq M$ the execution time $T(G, B)$ is:

$$T(G, B) = \begin{cases} T_1(G, B) = T_s * (p - 1) + T_c * N/(G*p) \text{ in } R_1 \\ T_2(G, B) T_s * (N/G - 1) + T_c \text{ in } R_2 \end{cases}$$

Fixed the number of processors p, the parameters β^I, β^E, β and τ are constants architectural dependent and t_0, t_1, t_{2i} , M and N are variables depending on the instance of the problem. The actual values for these variables are known at running time. An analytical expression for the values (G, B) leading to the minimum, will depend on the five variables and seems to be a very complicated problem to solve. Instead of an analytical approach we will approximate the values for (G, B) numerically. An important observation is that $T(G, B)$ first decreases and then increases if we keep G or B fixed and move along the other parameter. Since, for practical purposes, all we need is to give values for (G, B) leading us to the valley of the surface, a few numerical evaluations of the function $T(G, B)$ will be sufficient.

To introduce the model into a tool that automatically computes G and B, during the execution of the first band, the tool estimates the parameters defining the function $T(G, B)$ and carries out the evaluation of the optimal values of G and B. The overhead introduced is negligible, since only a few evaluations of the objective function are required. After this first test band, the execution of the parallel algorithm continues with the following bands making use of the optimal Grain and Buffer parameters.

4 Validation of the Model

We have applied the model to estimate the optimal grain *G* and optimal buffer *B* for the 0-1 Knapsack Problem (KP) [3] and the Resource Allocation Problem (RAP) [2]. A pipeline algorithm for the RAP has the property that *body2* does not take constant time. Table 1 presents the values for *(G-Model, B-Model)* obtained with the model, the *llp*-running time of the parallel algorithm for this parameters (*Real Time*), the running times obtained with the best values of *(G-Real, B-Real)* and the best running time (*Best Real Time*). The table also shows the error made ((*Best Real Time - Real Time*) / *Best Real Time*) when we consider the parameters provided by the tool instead of the optimal values.

The model shows an acceptable prediction in both examples with an error not greater than 15 %.

Table 1. Estimation of *G*, *B* for the KP and RAP.

	P	G-Model	B-Model	Real Time	G-Real	B-Real	Best Real Time	Error
KP	2	10	2048	140.08	20	5120	138.24	0.003
KP	4	10	768	70.84	20	1792	69.47	0.053
KP	8	10	512	35.85	20	768	35.08	0.097
KP	16	10	192	18.29	10	768	17.69	0.150
RAP	2	10	10	73.33	5	480	70.87	0.034
RAP	4	5	10	36.73	5	160	36.01	0.020
RAP	8	2	10	19.26	5	40	18.45	0.044
RAP	16	1	10	10.79	5	40	9.57	0.127

5 Conclussions

We have developed an analytical model that predicts the effects of the Grain of processes and Buffering of messages when mapping pipeline algorithms. The model allows an easy estimation of the parameters through a simple numerical approximation. The model is capable to be introduced into tools (like *llp*) that produce the optimal values for the Grain and Buffer automatically.

References

1. Andonov R., Rajopadhye S.. Optimal Orthogonal Tiling of 2D Iterations. Journal of Parallel and Distributed computing, 45 (2), (1997) 159-165.
2. Morales D., Almeida F., García F., González J., Roda J., Rodríguez C.. A Skeleton for Parallel Dynamic Programming. Euro-Par'99 Parallel Processing Lecture Notes in Computer Science, Vol. 1685. Springer-Verlag, (1999) 877–887.

3. Morales D., Roda J., Almeida F., Rodríguez C., García F.. Integral Knapsack Problems: Parallel Algorithms and their Implementations on Distributed Systems. Proceedings of the 1995 International Conference on Supercomputing. ACM Press. (1995) 218-226.
4. Ramanujam J., Sadayappan.. Tiling Multidimensional Iterations Spaces for Non Shared-Memory Machines. Supercomputing'91. (1991) 111-120.