The Tuning Problem on Pipelines[1]

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

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

Abstract.. Performance analysis and prediction is an important factor determining the efficiency of parallel programs. Considerable efforts have been made both in pure theoretical analysis and in practical automatic profiling. Unfortunately, contributions in one area seem to ignore the results of the other. We introduce a general performance prediction methodology based on the integration of analytical models and profiling tools. According to this approach we have developed a tool that automatically solves the prediction of the parameters for optimal executions of parallel pipeline algorithms. The accuracy of the proposal has been tested on a CRAY T3E for pipeline algorithms solving combinatorial optimization problems. The results obtained suggest that the technique could be successfully ported to other paradigms.

1 Introduction

One disappointing contrast in parallel systems is between the peak performance of the parallel systems and the actual performance of parallel applications. Performance prediction is important in achieving efficient execution of parallel programs, since it allows to avoid the coding and debugging cost of inefficient strategies. Most of the approaches to performance analysis fall into two categories: Analytical Modeling and Performance Profiling.

Analytical methods use models of the architecture and the algorithm to predict the program runtime and the analysis can be independent from the target architecture. Parallel performance analysis typically studies the sensitivity of application performance as a function of the system size and problem size. Profiling may be conducted on an existing parallel system to recognize current performance bottlenecks, correct them, and identify and prevent potential future performance problems.

Currently, the majority of performance metrics and tools devised for performance evaluation and tuning rarely provide predictions of performance for executions that have not been measured. Many projects have been developed to create trace files of

[1] The work described in this paper has been partially supported by the Spanish Ministry of Science and Technology (CICYT) TIC1999-0754-C03.

events with associated time stamps and then examine them in post-mortem fashion by interpreting them graphically. The ability to generate trace files automatically is an important component of many tools like PICL [2], Dimemas [3], Kpi [1].

Although much work has been developed in Analytical Modeling and in Parallel Profiling. We claim that to obtain automatic and effective practical tools with predictive ability, both fields should be integrated. When executing an algorithm, the user should know the analytical model and provide the complexity analytical formula of the algorithm implemented. According to this formula, the profiler could compute the parameters needed by this formula for the performance prediction and use them on the optimal execution of the algorithm.

In section 2 we formalize the General Tuning Problem and propose a generic methodology to approach it. In section 3 we apply this technique to the Pipeline Paradigm. We have extended the La Laguna Pipeline tool, *llp* [4], with automatic profiling facilities. The new system measures the relevant code sections, finds the constants involved, minimizes the complexity function and finds the optimal parameters for the execution on the current input and architecture. The feasibility of the technique and its accuracy has been contrasted over pipeline algorithms for combinatorial optimization problems.

2 The Problem and the Methodology

As is common in Computability Theory, any algorithm A determines a function F_A from an input domain D to an output domain. Usually the input domain D is a cartesian product $D = D_1 x ... x D_n$, where D_i is the domain for the *i-th* parameter

$$F_A : D = D_1 x ... x D_n \subset \Sigma^* \rightarrow \Sigma^*$$

such that $F_A(z)$ is the output value for the entry z belonging to D. This algorithm A, when executed with entry z on a machine M, spends a given *execution time*, denoted $Time_M(A(z))$. In most cases this $Time_M(A(z))$ function can be approximated by an analytical Complexity Time formula $CTime_M(A(z))$. We assume that $CTime_M(A(z))$ represents with enough accuracy the actual function time $Time_M(A(z))$.

We will classify the parameters $D_1 x ... x D_n$ of F_A into two categories $\mathbb{T}$ ($\mathbb{T}$ comes for tuning parameters) and I (for true Input parameters).

We define that $x \in D_i$ is a "*tuning parameter*", $x \in \mathbb{T}$ if and only if, occurs that x has only impact in the performance of the algorithm but not in its output. We can always reorder the tuning parameters of A, to be the first ones in the algorithm:

$$\mathbb{T} = D_1 x ... x D_k \text{ and } I = D_{k+1} x ... x D_n$$

With this convention is true that $F_A(x, z) = F_A(y, z)$ for any x and $y \in \mathbb{T}$, but, in general, $Time_M(A(x, z)) \neq Time_M(A(y, z))$.

The "Tuning Problem" is to find $x_0 \in \mathbb{T}$ such that,

$$CTime_M(A(x_0, z)) = min \{ CTime_M(A(x, z)) \ / x \in \mathbb{T} \} \ (1).$$

The general approach that we propose to solve the tuning problem is:

1. Profiling the execution to compute the parameters needed for the Complexity Time function $CTime_M(A(x, z))$.
2. Compute $x_0 \in \mathbb{T}$ such that minimizes the Complexity Time function $CTime_M(A(x, z))$.
3. At this point, the predictive ability of the Complexity Time function can be used to predict the execution time $Time_M(A(z))$ of an optimal execution or to execute the algorithm according to the tuning parameter $\mathbb{T}$.

3 The Pipeline Tuning Problem

We will restrict our study to the case where the code executed by every processor of the pipeline is the *M* iteration loop of figure 1. In the loop that we consider, *body0* and *body1* take constant time, while *body2* depends on the iteration of the loop. This loop represents a wide range of situations, as is the case of many parallel Dynamic Programming algorithms [4].

```
void f() {
  Compute(body0);
  i = 0;
  While(i < M) {
    Receive();
    Compute(body1);
    Send();
    Compute(body2, i);
    i++
  }
}
```

Fig. 1. Standard loop on a pipeline algorithm.

The virtual processes running this code must be assigned among the *p* available processors following a mixed block-cyclic mapping on a one way ring topology. The grain *G* of processes assigned to each processor is the second tuning parameter to ponder. Buffering data reduces the overhead in communications but can introduce delays between processors increasing the startup of the pipeline. The size *B* of the buffer is our third tuning parameter.

The Analytical Model and the Pipeline Tuning Solver

The optimal tuning parameters $(p_0, G_0, B_0) \in \mathbb{T} = \{ (p, G, B) / p \in \mathbb{N}, 1 \le G \le N/p, 1 \le B \le M\}$ must be calculated assumed that the constants characterizing the architecture and the constants associated to the algorithm have been provided. As the Analytical Model we will follow the general model presented in [4].

In this model, 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. An external reception is represented by (β^E) and an internal reception by (β^I), including the time spent in context switching. The variables t_0, t_1, t_{2i} respectively denote the times to compute *body0*, *body1* and *body2* at iteration *i*.

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

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

For a problem with N stages on the pipeline (N virtual processors) and a loop of size M (M iterations on the loop), the execution time is determined by:

$$T(p, G, B) = max \{ T_s * (p - 1) + T_c * N/(G*p), T_s * (N/G – 1) + T_c\}$$

with $1 \leq G \leq N/p$ and $1 \leq B \leq M$. $T_s * (p-1)$ holds the time to startup processor p and $T_c * N/(G*p)$ is the time invested in computations after the startup. The tuning parameter is $\mathbb{T} = (p, G, B)$ and the input parameter is $I = (N, M, t_0, t_1, t_2)$.

The La Laguna Pipeline tool, *llp*, enrolls a virtual pipeline into a simulation loop according to the mapping policy specified by the user. This policy is determined by the grain parameter, *G*. *llp* also provides a directive to pack the data produced on the external communications. The directive establishes the number of elements *B* to be buffered. We have instrumented *llp* to solve automatically the Pipeline Tuning Problem. The profiling step runs sequentially just one stage of the pipeline so that the whole set of input parameters is known in advance. The minimization function for the analytical model supplying the parameters for an optimal execution is then applied.

To solve the tuning problem the input parameters $I = (N, M, t_0, t_1, t_2)$ must be known before the minimization function be called. Given that N and M are provided by the user, (t_0, t_1, t_2) must be computed for each instance. Since the computations on the pipeline code (fig 1) are embedded into two *llp-communications* calls, during the profiling phase, these *llp* routines are empty and just introduce timers.

Computational Results

To contrast the accuracy of the model we have applied it to estimate (p_0, G_0, B_0) for the pipeline approach on the dynamic programming formulation of the knapsack problem (KP) and the resource allocation problem (RAP). The machine used is a CRAY T3E providing 16 processors. For the problems considered, we have developed a broad computational experience using *llp*. The computational experience has been focused to finding experimentally the values (p_0, G_0, B_0) on each problem. The tables denote the optimal experimental parameters as *G-Real*, *B-Real*. These were found by an exhaustive exploration of the *GxB* search space. *Best Real Time* denotes the corresponding optimal running time. The running time of the parallel algorithm for parameters (G_0, B_0) automatically calculated solving equation (1) is presented in column *Real Time*. The tables also show the error made ((*Best Real Time* - *Real Time*) / *Best Real Time*) by considering the parameters automatically provided by the tool. The very low error made with the prediction makes the technique suitable to be considered for other parallel paradigms.

Table 1. *(G_0, B_0)* prediction.

P	G_0	B_0	Real Time	G-Real	B-Real	Best Real Time	Error
Knapsack Problem (KP12800)							
2	10	3072	138.61	20	5120	138.24	0.0026
4	10	1536	70.69	20	1792	69.47	0.017
8	10	768	35.69	20	768	35.08	0.017
16	10	256	18.14	10	768	17.69	0.025
Resource Allocation Problem (RAP1000)							
2	2	10	74.62	5	480	70.87	0.053
4	2	10	37.74	5	160	36.01	0.048
8	2	10	19.26	5	40	18.45	0.044
16	2	10	10.06	5	40	9.76	0.031

4 Conclusions

We have presented a formal definition of the General Tuning Problem and proposed a generic methodology to approach it. A special case for Pipelines has been approached. We have extended the La Laguna Pipeline tool with automatic profiling facilities to solve the Tuning Problem for Pipelines. The feasibility of the technique and its accuracy has been contrasted on combinatorial optimization Problems on a CRAY T3E.

References

1. Espinosa A., Margalef T., Luque E.. Automatic Performance Evaluation of Parallel Programs. Proc. Of the 6th EUROMICRO Workshop on Parallel and Distributed Processing. IEEE CS. 1998. 43-49.

2 Geist A., Heath M., Peyton B., Worley P.. PICL: Aportable Instrumented Communications Lybrary, C Reference Manual. Technical Report TM-11130. Oak Ridge National Laboratory. 1990.

3. Labarta J., Girona S., Pillet V., Cortes T., Gregoris L.. Dip: A Parallel Program Development Environment. Europar 96. Lyon. August 1996.

4. Morales D., Almeida F., Moreno L. M., Rodríguez C.. Optimal Mapping of Pipeline Algorithms. EuroPar 2000. Munich. Sept. 2000. 320-324