Parasol I

Arjan de Mes, FWI, UoA

September 20, 1993

Introduction

For Parasol I a model needed to be created that could perform the following functions:

- Correctly represent existing ('real') machines.
- Allow user to change the parameters, thus creating virtual machines.
- Return requested parameters to the SAD evaluator.

These points are explained in more detail below.

1 Machine representation

To correctly represent a real machine in a model, we need to look at what influences the machines performance, e.g. a faster integer multiplication (usually) leads to faster execution of (most) programs.

The model presented here contains all parameters of the machine that (can) influence the performance. This can be split up into three sections by separating processor, topology and SPMD programming models. These points will be discussed later in this section.

Some data can be combined to form 'other data', giving a better insight into machine performance. Two levels can be distinguished: combinations of data at processor level and combinations of data at machine level.

- Interesting indexes at processor level are ratios between arithmetic performance and communication speed. Ratios against latency do not seem so interesting.
- Few indexes are interesting when one number represents a machine. [Markatos92] presents the sum of the maximum transfer rates of all the processors as a useful index for the communications performance. FLOPS en MIPS might be interesting for a sequential machine or a single processor, but not for parallel machines.

1.1 Processor representation

Processor parameters that influence performance are:

- arithmetic speed
- branch cost
- process management

- delay caused by communication

The *arithmetic speed* can be split up per numeric type (integer, long integer, float and double). For each type the timing per arithmetic function has to be specified. The assignment, compare and cast functions are also seen as arithmetic functions in this context.

The execution time of an instruction can vary. A multiplication by zero or one is likely to be completed faster than multiplication by e.g. 123. An average is taken. The fetch time of the instruction and its two operands may vary depending on whether they are present in cache memory or not. Here an average time is chosen. This average time is then combined with instruction and operand size and added to the time needed to execute the instruction. This result is the time returned to execute the instruction.

Due to this, memory timings do not need to be added to the model separately. Having separate memory timings would introduce several complications; it is extremely difficult to measure memory speed on a multiuser system (with caching) and it would be difficult for the SAD evaluation level to have to consider these timings.

The *branch cost* influences the performance when an instruction pipeline is present (see [Sarkar89]). The average delay caused by a branch by 'an average program' is returned.

The *process management* can introduce serious overhead when many processes are active or when process switching or process creation is expensive. Average timing for switching and creation need to be present in the model.

The *delay caused by communication* is usually very low. State of the art transfer rates are always realized by separate communication processors (using DMA), or by fast processors that would spend at most a few percent of the cycles communicating. Nevertheless delays caused by receiving and routing are taken into account. When a separate routing processor is present, delay caused by routing is likely to be 0%.

1.2 Topology representation

Topology parameters that influence performance are:

- processor interconnection
- connection speeds
- creation time for virtual topologies

In order to model a machine correctly, it is necessary to have information about the hardware processor interconnection, but also about how virtual *processor interconnection* is implemented on the hardware level. Processor interconnection is specified link by link. For each link timings (latency and transfer rate) and connected processors are specified. This style of specifying topologies is extremely elaborate but provides a system for defining any possible topology.

However: Benchmarking a real machine can be quite some work. Obtaining correct parameters for all the links needs to be done automatically. When doing this, one needs to store all the results. This is done in a datastructure, which can be written to disk. It is not necessary (nevertheless possible) to specify all the links by typing them.

Creating or changing to other virtual topologies can be quite an overhead. Although this is almost never done in SPMD programs, it is provided in this model for completeness.

1.3 SPMD model representation

Two SPMD supporting programming models are provided in this model:

- PVM
- Express

2 Interface with the SAD evaluator

To interface with the SAD evaluator, all the data with respect to the (possibly virtual) machine need to be present in the Parasol I layer. The data is present in two levels: on disk and in memory. The global idea is sketched in Figure 1.

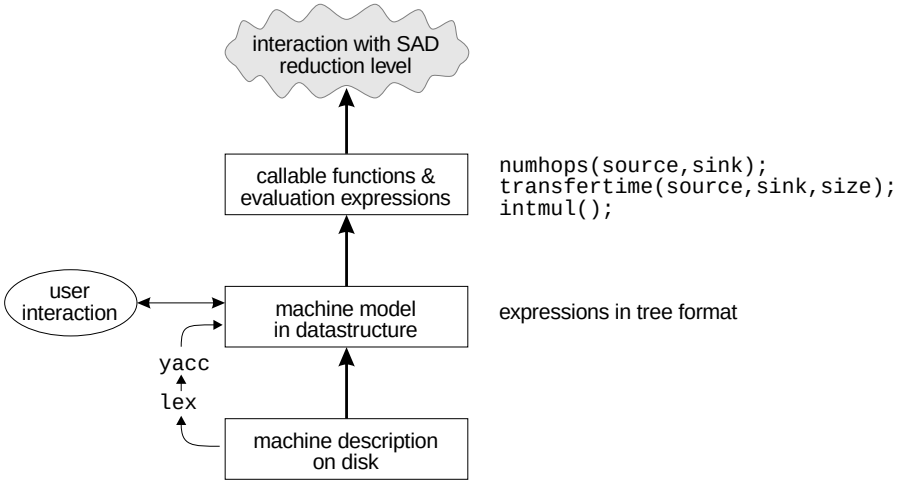

Figure 1: General overview for Parasol I

2.1 The SAD interface functions

It is at present unknown at which level the SAD evaluator will be requesting information on machine parameters, and in which form or method this information will be needed.

Current ideas on the matter suggest that the code returning the machine parameters is linked with the SAD evaluator level. The parameters will then be communicated through function calls.

Other possibilities, such as several message passing methods for UNIX machines, are subjects for future research.

2.2 The datastructure

The description of the machine is stored in memory in a hierarchial (tree-like) form. At the highest level a linked list of machines is stored. Each machine has four ‘children’. The reference to the processor description points to the first element of a linked list. The communication description is nothing more than a pointer to a list of topology descriptions. The references to the SPMD supporting models are combined and form the machine descriptions third child. This hierarchial structure is represented by Figure 2. The combined global machine information (fourth child) is not shown in Figure 2 because

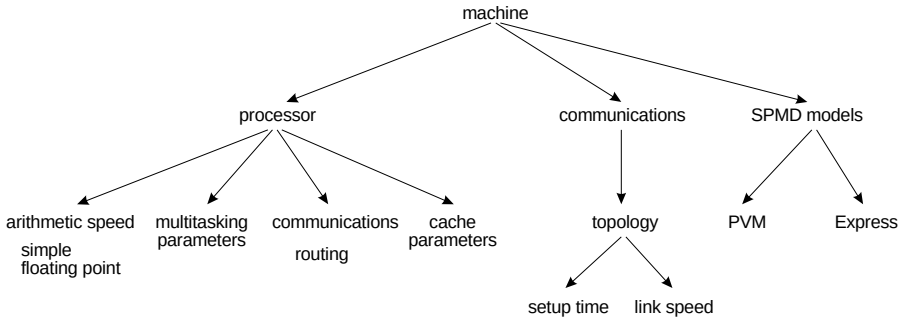

Figure 2: General overview for Parasol I

it is extremely straight-forward (see the introduction of Section 1). The maximum total transfer rate is stored in this fourth child (as a double precision floating point number).

For each processor parameter as `branchdelay` (see Section 1.1), `createjob`, etc. there is an entry in the structure describing the processor. Also there are four references to the arithmetic timings, one for every numeric type modelled. All timings are stored as double precision floating point numbers. The remaining two references are to the cache parameters and the combined processor data.

The topologies (as mentioned in Section 1.2) are stored in a very elaborate manner. For each link parameters are stored. These parameters will be used intensively in calculating global communication speeds, therefore they are stored in an array instead of a list. Next to this array of link descriptions, topology parameters such as the topology creation time are also stored here. A link description consists of two arrays (one for the processors it connects, the other to define the type of the processor), latency time, transfer speed, and a boolean to define if this is an I/O link.

The SPMD models are represented using expressions with references. References (as explained later in Section 2.4) have to be looked into in the very near future; this however does not influence the datastructure in which these expressions are stored. Expressions are stored in tree format. References will have a function-like form.

2.3 The disk format

The description of the machine as stored on disk can be divided into the four sections discussed in Section 1. These four sections have no specific order, but they must be preceded by a header indicating that the section has started. For example:

```
machinename = "A Virtual Machine"
processor = 1
    ....  (description for processor "1")
processor = 2
    ....  (description for processor "2")
topology = 100
    ....  (description for topology "100")
machinecombinations
    ....  (combined machine parameters)
modelsection = PVM
    ....  (performance params for PVM implementation)
```

```
modelsection = Express
    ....  (performance params for Express implementation)

machinename = "Another Virtual Machine"
 .
 .  (etc.)
 .
```

First the machine is named "A Virtual Machine". This is just for human convenience and doesn't influence the model. Then the description of the first processor starts. This processor will be referred to as processor '1'. This line is seen as a header for the processor definition. The processor description may contain the following definitions:

```
name = "Some Processor"
fp_type = floating
mantissa = 24           # number of bits in the mantissa
exponent = 8            # number of bits in the exponent
pipelinesize = 1        # instructions lost in branch
branchdelay = 20.0      # delay in nanoseconds
maxrecvdelay = 30.0     # delay in percentage, 0% = no delay
maxroutedelay = 5.0     # delay in percentage
createjob = 1.2e+3      # time in nanoseconds
jobswitch = 6e+2        # time in nanoseconds
integer_timings
long_integer_timings
float_timings
double_timings
processorcache
combinations
```

The latter 6 entries are headers for further subsections. The subsections define average times for the four numeric types; integer, long integer, float and double, and specify some cache parameters and the combined processor parameters. These subsections will not be discussed here.

Then another processor (referred to as processor '2') is defined. This is done in the same style as processor '1'. This section is followed by a topology specification. This topology will be referred to as topology '100'. Topologies may have the same numbers as processors, as they are of different types. The line is again seen as a header for the topology definition. The topology description may contain the following definitions:

```
toponame = "pipeline"
hardware = 1
create_time = 3.2e+3    # time in nanoseconds
num_links = 2
linknumber = 0
 .
 .  (etc.)
 .
```

The last entries are headers for link specification sections. In this example two links will be specified. One of these link specifications could be:

```
linknumber = 0          # copied from previous ...
linkio = 0              # yes,no,true,false,1,0
linklatency = 2.3e-1    # time in nanoseconds
linkspeed = 2667231     # number of bytes per second
procs_connected = 2     # 1 = I/O, >2 = e.g. Ethernet
processortype0 = 1      # refers to processor ID
processornumber = 43
processortype1 = 1      # refers to processor ID
processornumber = 45
```

Here processor 43 and processor 45 are interconnected by a link with a latency of 0.23 nanoseconds and a transfer rate of 2667231 bytes per second. Both processors are of type '1', which in this case would be 'Some Processor'. This link is not an I/O link.

This topology definition indirectly specifies the number of processors in the machine. If a link is connected to 'processor number 43', evidently processor 43 will be in the machine. All links are assumed to be full duplex. This reduces the amount of data needed to specify a topology and is also very intuitive. The 'real' machines we wish to represent with this model (Parsytec GCel and a SUN Sparc station pool) are suitable for this assumption.

The combined machine parameters are preceded by the header 'machinecombinations'. This section is very short (see Section 1). There is one possible entry:

```
maxtotalspeed = 1.1e+9  # number of bytes per second
```

The SPMD supporting programming models that may be specified are 'PVM' and 'Express'. Each has different parameters that can be / need to be specified. These parameters are all represented as expressions with references to other fields in the database. These references are explained in Section 2.4. That section refers to future research, so the file format can not be specified precisely at this point. However for the PVM definition the following PVM primitives will be described: `spawn`, `sync`, `broadcast`, `multicast`, `send`, `receive`, `packbyte`, `packdouble`, `packfloat`, `packint`, `packlong`, `packstring`, `unpackbyte`, `unpackdouble`, `unpackfloat`, `unpackint`, `unpacklong` and `unpackstring`. For Express the following primitives will be described: `sync`, `broadcast`, `multicast`, `send`, `receive`, `exchange`, `combine` and `concat`.

The reason these parameters are all specified, in references to the other fields and parameters, is they are of greater complexity and depend entirely on external parameters. The part that is internalized by these expressions is the implementation of the models on the machine.

2.4 Referring to other fields and parameters

To correctly represent time responses for SPMD programming models the source for these functions will need to be examined and split up into an expression referring only to primitive functions, whose timings are specified elsewhere in the machine description.

The execution time for these functions becomes an expression depending on the parameters to the function and the timing of primitive functions for the machine.

How this will be represented on disk and in the datastructure is material for future research.

References

[Andrews91] J. B. Andrews and C. D. Polychronopoulos. *An Analytical Approach to Performance / Cost Modeling of Parallel Computers*, Center for Supercomputing Research and Development University of Illinois at Urbana-Champaign, Journal of Parallel and Distributed Computing, 1991.

[Dimpsey91] R. T. Dimpsey and R. K. Iyer. *Performance Prediction and Tuning on a Multiprocessor*, Center for reliable and high-performance computing, University of Illinois at Urbana-Champaign, ACM, 1991.

[Markatos92] E. P. Markatos and T. J. LeBlanc. *Shared-Multiprocessor Trends and the Implications for Parallel Program Performance*, University of Rochester, Computer Science Department, Rochester, New York, May 1992.

[Muller93] H. Muller. *Simulating Computer Architectures*, Faculty of Mathematics and Computer Science, University of Amsterdam, Amsterdam, 1993.

[Ronde93] J. F. de Ronde. *Position Paper Parasol I & II CAMAS*, Faculty of Mathematics and Computer Science, University of Amsterdam, Amsterdam, 1993.

[Rosen76] S. Rosen. *Lectures on the Measurement and Evaluation of the Performance of Computing Systems*, Purdue University, Philadelphia, Pennsylvania, SIAM, 1976.

[Sarkar89] V. Sarkar. *Determining Average Program Execution Times and Their Variance*, IBM Research, Watson Research Center, Yorktown Heights, New York, ACM, 1989.

[Sunderam89] V. S. Sunderam. *PVM: A Framework for Parallel Distributed Computing*, Department of Math and Computer Science, Emory University, Atlanta, Concurrency: Practice and Experience, 1989.

[Tanenbaum88] A. S. Tanenbaum. *Computer Networks*, Prentice Hall, Englewood Cliffs, New Jersey, 1988.