Cost Evaluation from Specifications for BSP Programs

Virginia Niculescu

Babeş-Bolyai University
Dept. of Computer Science
Cluj-Napoca, 400084 ROMANIA
vniculescu@cs.ubbcluj.ro

Abstract

BSP has shown that structured parallel programming is not only a performance win, but it is also a program construction win, especially if we add a formal method for designing. Maybe the most important advantage that BSP brings is the effective cost model that allows a good evaluation of the performance. The paper presents a technique for cost evaluation from specifications for BSP programs. We consider parameterized specifications and processes for BSP programs, and the parameters are the number of processes, the index of the local process, and the data distribution. The possibility of counting the number of communications from postconditions, allows us to make a cost evaluation even at the early stages of the design, and so it leads us to the right decisions.

1. Introduction

To become effective, parallel computation requires at least three distinct components to be in place. The first is the parallel hardware that will execute parallel applications; the second is the abstract machine, or programming model, in which parallel applications will be written; and the third is the software design process that allows applications to be built from specifications [7].

One of the possible choices for a general-purpose parallel programming model is BSP – *Bulk Synchronous Parallelism*. A very important advantage of BSP is that its cost measures give the real cost of a program on any architecture. Because it separates communication from synchronization, it is particularly clean and simple. This separation allows us to develop a simple and formal software construction process for it.

In order to satisfy the third requirement for an effective parallel computing method, a method based on parameterized specifications and formal derivation could be used [5]. Starting from these parameterized specifications it is possible to evaluate communication costs from postconditions, before developing the program. This way we may choose the right distribution - the one who leads to the lowest costs – without developing the programming for each distribution.

2. Bulk Synchronous Parallelism

The BSP model was proposed by Valiant [8] as a "bridging model" that provides a standard interface between parallel architectures and algorithms. A BSP computer contains a set of processor-memory pairs, a communication network allowing inter-processor delivery of messages, and a global synchronization unit that executes collective requests for a synchronization barrier. The computation is divided into *supersteps* separated by global synchronization steps, and packets sent in one superstep are assumed to be delivered at the beginning of the next superstep.

The properties of architectures are captured by four parameters. These are: the raw speed of the machine (which can be ignored by expressing the remaining parameters in its units), the number of processors – p, the time required to synchronize all processors – l *latency*, and the ability of the network to deliver messages under continuous load – g *gap* (which reflects network bandwidth on a per processor basis).

The BSP model ignores the particular topology of the underlying machine; this rules out any use of network locality in algorithm design. Thus, the model only considers two levels of locality, local (inside the processor) and remote (outside a processor), with remote access usually being more expensive than local ones.

1-4244-0054-6/06/$20.00

The cost of a superstep is given by:

$$cost = w + hg + l$$

where w is the maximum local computation in any processors during the superstep, h is the maximum number of global communications into or out of any processor during the superstep.

The organization of programs as a sequence of steps reduces the complexity of arranging communication and synchronization. This makes it straightforward to extend techniques for constructing sequential programs to BSP programs.

3. Formal Derivation of BSP Programs

We use a derivational method that considers a parallel program as a number of cooperating *parameterized* processes with similar structures.

A parallel program is considered to be formed of many parameterized processes $S.q(0 \leq q < p)$, which are running in parallel. There is no shared memory, and point-to-point communication is considered. Since in BSP model, a parallel machine consists of a set of processors, each with its own private memory, and an interconnection network that can route packets between processors, we may reason about BSP programs as a set of parameterized processes that communicate via message-passing.

A parameterized process is much like a procedure in sequential programming. The difference is that, instead of having only one instantiation in a sequential program, we have many instantiations in a parallel program.

In sequential programming the Hoare-triple [3]

$$\{Q\}S\{R\}$$

is commonly used to denote a formal specification of a program S. This notation expresses that if the program S starts in a state described by the predicate Q, and the program terminates, then, upon completion, the predicate R is satisfied (partial correctness).

For specifying a parallel program, both pre- and post-conditions, Q and R, are split up as conjunctions of $p(p > 0)$ local pre- and post-conditions, and a process is associated with each such pair:

$$\{Q.q\}S.q\{R.q\}, \forall q : 0 \leq q < p.$$

A parameterized specification refers local variables, the number of processes (p), the index of the local process (q), and also elements of the distributed objects which represent the input of the problem. These elements of the distributed objects are considered only in specifications and they cannot be modified. Therefore, *the parameters of a specification* are q, p, and a data distribution D. Many choices are possible for D, each of them having an impact on the complexity of the parallel program.

Such a specification forms the starting point for a parallel program derivation, a formal construction of parameterized processes constituting a parallel program.

Our approach for obtaining a parameterized process S from a functional specification is similar to the methods used in sequential programming. We may use the classic rules for derivation from sequential programming [1, 3], and new rules for parallel composition and communication [2]. The classic method that obtains an invariant from a specification, in a calculational style, may be used.

4. Distributions

Data distributions have a serious impact on time complexity of parallel programs developed based on domain decomposition, which are very conveniently implemented using BSP model. Distributions are considered to be parameters of our programs, and they have to be carefully analyzed since they may considerably change the complexity of our programs. Simple static distributions are going to be considered; in a static distribution the assignments are not changed during the execution of the program. The simple distributions are characterized by:

- the number of data elements assigned to one process;
- the data distribution on processes.

If the data input is a vector, the simple distribution is called one-dimensional distribution, and if it is a multi-dimensional array, the distribution is called Cartesian distribution [4]. Set-distributions [6] may also be considered. Using them we may distribute one date to a set of processors.

We use $\overline{n}$ to denote the set $\{\forall i : 0 \leq i < n : i\}$.

Definition 1 *$D = (\delta, A, B)$ is called a (one-dimensional) distribution if A and B are finite sets, and δ is a mapping from A to B; set A specifies the set of data objects (an array with n elements that represent the indices of data objects), and the set B specifies the set of processes, which is usually $\overline{p}$. The function δ assigns each index $i(0 \leq i < n)$, and its corresponding element, to a process number.*

Well-known ways of distributing an array are: every element to one unique process (*identity*), assigning p

equally-sized consecutive array segments (*linear*), and assigning elements cyclically (*cyclic*).

Distributions of multi-dimensional arrays may be modeled by Cartesian distributions. In what follows, it is assumed that an $m \times n$ matrix is distributed across processes.

Definition 2 *A Cartesian distribution is defined by a Cartesian product of one-dimensional distributions. The Cartesian product of two one-dimensional distributions* $D0 = (\delta 0, \overline{m}, \overline{M})$, $D1 = (\delta 1, \overline{n}, \overline{N})$ *is defined by:*

$$D0 \times D1 = (\delta 0 \times \delta 1, \overline{m} \times \overline{n}, \overline{M} \times \overline{N})$$

where the function $\delta 0 \times \delta 1$ *assigns a pair of process numbers to each array index pair.*

Formally written, we have $\delta 0 \times \delta 1 = (\lambda i, j \cdot (\delta 0.i, \delta 1.j))$.

The Cartesian product of two one-dimensional distributions uses a process pair as identification for a process. Cartesian distributions of matrices can be obtained by distributing the rows of the matrix independently from the columns. Since the processes number p is set, we can consider all decomposition such that $p = M * N$.

Two well-known examples of Cartesian distributions are $grid = cyclic^2$ and $block = linear^2$.

5. Cost Evaluation

BSP model gives us a cost model that is both tractable and accurate, and can be used as a part of the design process. This formal methodology for BSP program construction allows us to formally evaluate the costs based on the local postconditions.

The distributions determine how the global postcondition is split up, and the local postconditions determine the number of communications and the computational work of each process. Given a program's postcondition and a distribution, an evaluation of the cost, before developing the program, is possible.

We consider that we have p processes and a data distribution $\delta : \overline{n} \to \overline{p}$. The program's postcondition is split up into p local postconditions according to the distribution δ. For each local postcondition a process, establishing it, is created and associated to.

Under the hypothesis that every datum is assigned to one unique process, the total number of postconditions that refer to a particular datum is a measure of the number of communications of that datum.

For the datum e, the quantity $NOcc.e$ is introduced:

$NOcc.e =$
the no. of local postconditions in which e occurs.

For a BSP program, we are interested in finding the number h, and this depends on the fan_in and fan_out numbers of each process q. The fan_out number of process q may be computed based on $NOcc$ of each datum that is assigned to process q:

$$fan\_out_q = (\sum i : 0 \le i < n \wedge \delta.i = q : NOcc.e_i - A.e_i)$$

where

$$A.e = \begin{cases} 1, \text{ if } & e \text{ occurs in the postcondition} \\ & \text{of the process that contain it;} \\ 0, \text{ otherwise.} \end{cases}$$

The number $A.e$ is usually equal to 1, which means the process that contains a date e uses it in the local computation; but there are, however, exceptions to this, and Example 2 illustrates one such exception.

The fan_in number of process q can also be evaluated by counting all the data that are used in the local postcondition of process q, which are not assigned to process q.

$$fan\_in_q = (\sum i : 0 \le i < n \wedge \delta.i \neq q \wedge e_i \text{ occurs in } R.q : 1)$$

From these, we have $h_q = \max(fan\_in_q, fan\_out_q)$ (or $h = fan\_in_q + fan\_out_q$), and then $h = (\max\, q : 0 \le q < p : h_q)$ may be computed.

If the distribution is well balanced – perfect or homogenous – and the postconditions defined based on these distributions use the same amount of data, we may evaluate these numbers in a much simpler way. The total number of communication $NComm$ may be computed, and then h is computed by the formula $h = NComm/p$, if $h = \max(h_{in}, h_{out})$, or by the formula $h = 2NComm/p$, if $h = h_{in} + h_{out}$.

By summing over all data e, the total number of communications $NCom$ will be obtained from $NOcc.e$:

$$NCom = \left(\sum e :: NOcc.e - A.e\right)$$

The value of h is only determined by the way the program's postcondition is split up, and the distribution that is used.

This technique of counting communications allows a comparison of the distributions on the basis of their communication overhead. The applicability of the technique is not possible when common subexpressions exist; we have also considered that we need only one superstep for satisfying the postcondition. But, generally, in programs construction stepwise refinement is used, and we may apply this technique to partial postconditions. So, this technique can give us a fair

approximation of the communication overhead. This technique also allows us to choose the most appropriate distribution before developing the program.

The technique suits very well to BSP programs since the results that can be obtained are independent of the inter-communication network.

To illustrate this technique, an example for matrix multiplication is given. Two variants based on different ways of distributing data are discussed.

Example 1 [Matrix multiplication] Let us consider two matrices A and B of dimensions $m \times o$, and $o \times n$, respectively. Our goal is to compute the $m \times n$ matrix C, satisfying postcondition R:

$$R: \ C = A \ \times \ B.$$

We consider $p = M * N$ processes, each process being identified by an ordered pair $(s,t), 0 \leq s < M, 0 \leq t < N$. For the matrix C, a Cartesian distribution $D0 \times D1 (D0 = (\delta_0, \overline{m}, \overline{M}), D1 = (\delta_1, \overline{n}, \overline{N}))$is used, and we consider the following conditions hold: $M < o, N < o, M < m, N < n$. The matrix A is distributed using a Cartezian distribution $D0 \times D2 (D2 = (\delta_2, \overline{o}, \overline{N}))$, and the matrix B is distributed using a Cartezian distribution $D3 \times D1 (D3 = (\delta_3, \overline{o}, \overline{M}))$. We assume that these distributions are homogenous.

The local postconditon $R.s.t$ is:

$$\begin{array}{l} R.s.t: \\ \qquad (\forall i,j : 0 \leq i < m \wedge 0 \leq j < n \\ \qquad\qquad \wedge \delta 0.i = s \wedge \delta 1.j = t : \\ \qquad\quad c(i,j) = (\sum k : 0 \leq k < o : a(i,k) * b(k,j)) \end{array}$$

Note that $(\forall s,t : \ 0 \leq s < M \wedge 0 \leq t < N : \ R.s.t) \Rightarrow R$

In order to count the number of communications, quantities $NOcc.a(i,k)$ and $NOcc.b(k,j)$ are calculated. We assume that $\delta 0$ and $\delta 1$ are surjective.

$$\begin{array}{ll} & NOcc.a(i,k) \\ = & \quad \{\text{definition NOcc}\} \\ & |(\forall s,t : 0 \leq s < M \wedge 0 \leq t < N \wedge \delta 0.i = s \wedge \\ & \qquad (\exists j :: \delta 1.j = t) : (s,t))| \\ = & \quad \{\text{calculus, } \delta 1 \text{ is surjective }\} \\ & N * |(\forall s : 0 \leq s < M \ \wedge \delta 0.i = s : s)| \\ = & \quad \{\delta 0 \text{ is a function}\} \\ & N \end{array}$$

Similarly $NOcc.b(i,k) = M$.

Because we assume that the distributions are homogenous, we compute the total number of communications, and after that the number h.

$$\begin{array}{ll} & NCom \\ = & \{\text{definition of } NCom\} \\ & (\sum i,k : 0 \leq i < m \wedge 0 \leq k < o : \\ & \qquad NOcc.a(i,k) - 1) \\ & + \\ & (\sum k,j : 0 \leq k < o \wedge 0 \leq j < n : \\ & \qquad NOcc.b(k,j) - 1) \\ = & \{\text{calculus}\} \\ & om(N-1) + on(M-1) \end{array}$$

Then we have:

$$h = (om(N-1) + on(M-1))/(MN)$$

The cost of the program can then be evaluated as follows:

$$\begin{array}{ll} & cost \\ = & \ \{\text{definition}\} \\ & w + hg + l \\ = & \ \{\text{calculus}\} \\ & w + l + g(om(N-1) + on(M-1))/(MN) \end{array}$$

where w is the maximum local computation that can also be evaluated from the parameterized postcondition, $w = (m/M) * (n/N) * o$.

Remarks:

- The number h is independent of particular choices of $\delta 0$, $\delta 1$, $\delta 2$, and $\delta 3$.
- It is possible to determine M and N, $p = MN$ such that h is minimal. All possible values (M,N) are integer points on the hyperbola $p = M * N, 1 \leq M, N \leq p$, and the values of $NCom$ for fixed m, n, o lie on the line with a slope dependent on $\frac{m}{n}$. Hence, the minimal value for h depends on the ratio $\frac{m}{n}$ and in particular for $m = n$, h has a minimal value if p is square.

Example 2 [Matrix multiplication] We consider the same problem, but with matrix A distributed using a distribution $D0 \times D2 (D0 = (\delta_0, \overline{m}, \overline{M}), D2 = (\delta_2, \overline{o}, \overline{N}))$, and we distribute the transpose of matrix B, using a distribution $D3 \times D2 (D3 = (\delta_3, \overline{n}, \overline{M}))$ $(M < m, M < n, N < n, N < o)$.

The local postconditions are the same as in the first case. But, now we rewrite the postconditions in a dif-

ferent way:

$$
\begin{array}{l}
\quad (\forall i,j : 0 \le i < m \wedge 0 \le j < n \wedge \delta 0.i = s \wedge \delta 1.j = t : \\
\qquad c(i,j) = (\sum k : 0 \le k < o : a(i,k) * b(k,j)) \\
= \{\text{calculus}\} \\
\quad (\forall i,j : 0 \le i < m \wedge \delta 0.i = s \wedge 0 \le j < n \wedge \delta 1.j = t : \\
\quad\; c(i,j) = (\sum v : 0 \le v < N : \\
\quad\quad (\sum k : 0 \le k < o \wedge \delta 2.k = v : a(i,k) * b(k,j)))) \\
= \{w(i,j,v) \stackrel{not}{=} \\
\qquad (\sum k : 0 \le k < o \wedge \delta 2.k = v : a(i,k) * b(k,j))\} \\
\quad (\forall i,j : 0 \le i < m \wedge \delta 0.i = s \wedge 0 \le j < n \wedge \delta 1.j = t : \\
\qquad c(i,j) = (\sum v : 0 \le v < N : w(i,j,v)))
\end{array}
$$

Therefore, the program has two stages: the first for the computation of $w(i,j,v)$ values, and the second that combines these values.

The local postcondition for the first superstep is:

$$
\begin{array}{l}
R0.s.t : \\
\qquad (\forall i,j : 0 \le i < m \wedge \delta 0.i = s \wedge 0 \le j < n : \\
\qquad\; w(i,j,t) = (\sum k : 0 \le k < o \wedge \delta 2.k = t : \\
\qquad\qquad a(i,k) * b(k,j))).
\end{array}
$$

The element $a(i,k)$ only appears in the postcondition of the process containing it, and so, $NOcc.a(i,k) = 1, A.a(i,k) = 1$. The element $b(k,j)$ appears in M postconditions: $NOcc.b(k,j) = M$, $A.b(k,j) = 1$.

$$
\begin{array}{ll}
 & NOcc.b(k,j) \\
= & \{\text{definition}\} \\
 & |(\forall s,t : 0 \le s < M \wedge 0 \le t < N \wedge \delta 2.k = t \wedge \\
 & \qquad (\exists i :: \delta 0.i = s) : (s,t))| \\
= & \{\delta 0 \text{ is surjective}\} \\
 & M
\end{array}
$$

Hence, we have

$$
\begin{array}{l}
NCom_0 = on(M-1), \text{and} \\
h_0 = on(M-1)/(MN).
\end{array}
$$

For the second superstep the postcondition is:

$$
\begin{array}{l}
R1.s.t : \\
\qquad (\forall i,j : 0 \le i < m \wedge 0 \le j < n \\
\qquad\quad \wedge\ \delta 0.i = s \wedge \delta 1.j = t : \\
\qquad\; c(i,j) = (\sum v : 0 \le v < N : w(i,j,v)))
\end{array}
$$

We have $NOcc.w(i,j,v) = 1$. In order to compute $NCom_1$ and h_1 corresponding to the second superstep, we first have to establish $A.w(i,j,v)$:

$$
A.w(i,j,v) = \begin{cases} 1, & \text{if } v = \delta 1.j \\ 0, & \text{if } v \neq \delta 1.j \end{cases}
$$

Therefore

$$
\begin{array}{l}
NCom_1 = mn(N-1), \text{ and} \\
h_1 = mn(N-1)/(MN).
\end{array}
$$

Thus, we obtain the following result for the whole program:

$$
\begin{array}{ll}
 & cost \\
= & \\
 & w_0 + w_1 + (h_0 + h_1)g + 2l \\
= & \{\text{calculus}\} \\
 & w + g(on(M-1) + mn(N-1))/(MN) + 2l
\end{array}
$$

where

$$
\begin{array}{l}
w_0 = (mno)/(MN) \\
w_1 = (mnN)/(MN) \\
w = mn(o+N)/(MN)
\end{array}
$$

Remarks:

- Choosing the best distribution from the communication point of view depends on the values of n and o.
- The second variant needs two supersteps, so the latency l appears in the cost with the factor 2.
- The computation work is greater for the second variant, since the partial values $w(i,j,t)$ have to be summed in the second superstep.
- If we compare the results of these two examples for matrix multiplication, we may conclude that the first variant is generally better. The second variant is better only if $((o-n)m(N-1)g - mnN)/(MN) > l$; these are the cases when o is much greater than n.

6. Conclusions

BSP has shown that structured parallel programming is not only a performance win, but it is also a program construction win, especially if we add a formal method for designing.

BSP model proved to be very appropriate for problems with regular structure, and so, for problems based on domain decomposition. Using parameterized specifications we can take into account the data-distribution, even at the beginning of the construction process.

We have presented here a modality of evaluating costs of BSP programs from postconditions. This allows us to make a cost evaluation at the early stages of the design, and thus, we may choose, before developing the program, the data distributions that decrease the costs.

References

[1] E. Dijkstra. *A Discipline of Programming.* Prentice-Hall, Englewood Cliffs, NJ, 1976.

[2] D. G. G. M. Levin. A Proof Technique for Communicating Sequential Processes. *Acta Informatica*, 15:281–302, 1981.

[3] C. Hoare. An Axiomatic Basis for Computer Programming. *Communications of the ACM*, 12(10):576–580, 1969.

[4] L. Loyens. *A Design Method for Parallel Programs*. PhD thesis, Technische Universiteit Eindhoven, 1992.

[5] V. Niculescu. A Software Development Methodology for BSP Model. to be published.

[6] V. Niculescu. On Data Distribution in the Construction of Parallel Programs. *The Journal of Supercomputing*, 29(1):5–25, July 2004.

[7] D. Skillicorn and D. Talia. Models and Languages for Parallel Computation. *ACM Computer Surveys*, 30(2):123–136, June 1998.

[8] L. Valiant. A Bridging Model for Parallel Computation. *Communication of the ACM*, 33(8):103–111, August 1990.