Evaluation of Two BSP Libraries through Parallel Sorting on Clusters

Christophe Cérin
Université de Picardie Jules Verne,
LaRIA, Bat CURI, 5 rue du moulin neuf, 80000 AMIENS - France
cerin@laria.u-picardie.fr

Jean-Luc Gaudiot
University of Southern California
Electrical Engineering-Systems
3740 McClintock Ave.
Los Angeles, CA 90089-2563
gaudiot@usc.edu *

April 22, 2000

Abstract

We present our experiences in developping and tuning the performance at the user level, of (in core) parallel sorting on homogeneous and non homogeneous clusters with the use of the two available BSP (Bulk Synchronous Parallel model) libraries: BSPLib from Oxford university (UK) and PUB7 from the university of Paderborn (Germany). The paper is mainly about the communication performances of these two libraries and, in more general terms, it compares and summarizes the programming facilities and differences between them.
Keywords: Application results, performance analysis, BSP programming language

1 Introduction and motivations

Sorting in parallel records whose keys come from a linearly ordered set has been studied for many years[1]. The advent of parallel processing, in particular in the context of cluster computing, is pushed (boosted) by technology and maxims, for instance, one of them say that the processing time doubles each 18 months or that remote Read/Write in RAM becomes faster than local Read/Write with the use of new protocols and network interfaces. Therefore, we need faster sorts every 18 months build on the top of the technology. But studies on parallel sorting algorithms are also guided by fundamental questions about the properties of the inputs and outputs in order to better capture new architectural paradigms.

The organization of the paper is as follows. In section 2 we review some techniques based on sampling for sorting in parallel. In section 3 we recall the PSRS technique that we have implemented on different clusters (PC based, Alpha processors based, Fast Ethernet based,

*The work reported in this paper is supported in part by NSF Grants #MIP 9707125 and #INT 9602937

[1] See our web link at `http://www.laria.u-picardie.fr/~cerin/=paladin/` for a bibliography and a review of techniques about parallel sorting

Myrinet based) for sorting and the properties of the sort. At last, in the same section 3 we define our expectations. Section 4 concludes the paper.

2 Related work on sorting

It is now well understood that two generic approaches for sorting in parallel are of particular interest and work in practice (implemented algorithms are efficient on a variety of multiprocessor architectures):

MERGE-BASED: for this kind of algorithms, the different steps are summarized as follows: (1) each processor contains a portion of the list to be sorted (2) sort the portions and exchange them among all the processors (3) merge portions in one or many steps;

QUICKSORT-BASED: for this kind of algorithms, the different steps are summarized as follows: (1) the unsorted list is partitioned into a number of progressively smaller sublists defined by selected pivots (2) sort the sublists for which processors are responsible

Only a MERGE-BASED algorithm is under concern in our project, principally because it seems to be more easier to get results with the technique. We have experimented in [1] with such technique for stable[2] algorithms and with the Bulk Synchronous Parallel (BSP) toolkit as the target programming tool. Parallel Sorting by Regular Sampling [2] or PSRS for short is one technique in the category of one step merging algorithms. It is as follows:

Step 1: Perform a local sort; each processor selects p samples which are gathered onto process zero;

Step 2: Perform a local sort of p^2 samples; pick $p-1$ regular pivots k from the sorted p^2 samples; (pivots are picked at $ip + p/2, (1 \leq i \leq (p-1)$ intervals); broadcast these pivot values to all the processors from processor zero.

Step 3: Each processor produces p partitions of its local block using the $p-1$ pivots; each processor sends its partition i (marked by pivots k and k_{i+1} to processor i;

Step 4: Perform a merge of all the received partitions;

3 PSRS: a technique that match our expectations

In this section we explain why the PSRS (Parallel Sorting by Regular Sampling) technique is good for benchmarking, why we have chosen BSP as the programming target language and what are our expectations.

3.1 The properties of PSRS

In our research project we consider only *one step merge-based algorithms*. Such algorithms have low communication cost because they move each element at most once (and at the "right place"). Their main drawback is that they have poor load balancing if we don't care about it: it is difficult to derive a bound to partition data into sublists of "equal sizes". The case

[2]A sorting algorithm is *stable* if equal keys appear in the output buffer in the same order as in the input buffer.

of PSRS is more friendly because under the assumption that $n > p^3$, we can bound by $2.n/p$ the amount of data that a processor has to merge in the worst case and in the last step of the algorithm on a p processors machine (n denotes the input size, as usual). The constant 2 is excellent. Consequently, a very short round trip of experiments (less than 30) will be required to ensure that the measured execution times in different input sizes (but equal) are very closed: the variance of the execution times is good. So, we can benchmark in practice our system with this sort in a short time.

Theoretically speaking, it can be shown that the computational cost of PSRS can match the optimal $\mathcal{O}(n/p \log n)$ bound. To obtain this result we must ensure that all the data are unique. In the case of no duplicates, PSRS *guarantees* to balance the work within a factor of two of optimal in theory, regardless of the value distribution. In practive we observe a few percent of optimal. Authors in [3] addressed the effect of duplicate keys and they show that, in practice, it is not a concern. Further analysis in the paper shows that the presence of duplicates increases the upper bound on load balancing *linearly*: if d represents the key with the most number of duplicates and $U = 2.n/p$ the upper bound, then the upper bound with d duplicates becomes $U + d$. Practically speaking, the problem is to find the value of d such that the $2.n/p$ term does not dominate the d term; asymptotically speaking it is about $\mathcal{O}(n/p)$ times. One of our benchmark allow to observe this phenomenon. The question that arise is: "if we transform (by any mean) the input data such that now the inputs can be considered as unique, does the transformation cost influences the sorting time?". The question is treated in [1] and we developped special PUB codes for these "extremal cases".

The other important property of the PSRS technique is that during the communication step, the maximal amount of data communicated by a processor is at most n/p thus incurring a BSP communication cost of $g.\frac{n}{p}$ (g is the ability of a network to deliver messages under continuous traffic).

Furthermore, other approaches to load balancing (such as the Li and Sevcik algorithm [4]) can potentially handle nodes that do not make uniform progress. However in this paper [4], authors recognize that "the sublist[3] expansion decreases as s increases. However, average sublist expansion for $p \geq 64$ is still around 1.3 even when s is high as 128" (s is a parameter related to the number of pivots and p is the processor number). This means that some processors receive more or less 25% of work in supplement to the mean: it is quite big and a serious handicap against PSRS. We know by experience that PSRS is much better than 25% and it is about a few percents, below two percents. So, we have a negative "a priori" in using the work of Li and Sevcik for sorting on non homogeneous network and we hope that a "flavour like PSRS algorithm" could preserve the good properties of PSRS in sorting with non homogeneous processors.

It should be notice here that Li and Sevcik gave some pragmatics in order to calibrate the two main parameters of their model. They have suggested to set the overpartitioning parameter k to $\log p$ (p is the processor number) and to set the oversampling ratio s to 4. In doing this they obtain good load expansion metrics but they do dot provide any proof that it works for any p. The problem is that with different processor speeds, k, s parameters would have to be computed for each speed. The problem is open.

Moreover Li and Sevcik in [4] shows their experimental results on KSR1 machines and they also provides experimental results for the PSRS algorithm but not for the KSR1 but for TC-2000 and iPSC/860 machines. The comparisons are difficult to obtain. Note also that all the systems are not clusters made with the current technology.

At least, one can remark that the problem of sorting with duplicates is not explored in the work of Li and Sevcik [4]. It is a disadvantage against PSRS which is very stable with duplicates as we have seen below. We can consider that PSRS is more general than the work

[3]The sublist expansion is defined as the ratio of the maximum sublist size to the mean sublist size in [5]

of Li and Sevcik.

Until recently processors that sort in parallel according to the PSRS philosophy began by sequentially sorting their portions in the first phase of the algorithm and used regular sampling to select pivots. Even Shi and Schaeffer, the inventors of the PSRS technique in the original paper [2] said that "It appears to be a difficult problem to find pivots that partition the data to be sorted into ordered subsets of equal size without sorting the data first". In fact, an optimization can be done to bypass the first sorting phase if we concentrate (see [6]) on the notion of *quantiles*. Quantiles are a set of 'cut points' that divide a sample of data into groups containing (as far as possible) equal numbers of observations. The *p-quantiles* of an ordered set A of size n are the $p-1$ elements of rank $n/p, 2n/p, \cdots, (p-1)n/p$, splitting A into p parts of equal size. The *p-quantiles search problem* consists in determining those splitting elements of A. For instance, the problem is related to the problem of *finding the 'best'* (the maximum) as follows: the median - the $\lceil n/2 \rceil$th best - is given by a call to `find(t,0,n-1,n/2)` $(n = 2^k)$ where `find` is the procedure that find the index of the ith largest element of vector t between indexes 0 and $n-1$. To sum up, the randomized *find* algorithm that we have implemented is $\mathcal{O}(n)$ on average. Moreover, in practice we have that the number of required pivots is not of order n, but is related to the number of processors which is very inferior to the input size, we can expect near a 'linear time' (with a small constant) algorithm to select pivots which is better than a sequential sort in $\mathcal{O}(n \log n)$.

In other words, when implementing step 1 and step 4 of PSRS [2] we are faced to two choices if we want to be:

conform to the original paper . In this case we have to implement step 1 with a quicksort, then step 4 with a merge; This is done in the implementation of Sujithan for BSPLib;

not to be conform to the original paper , but conform to the spirit of the original paper (implementation choice). In this case, we implement step 1 with a quantile search (that do not offer ordered sequences after processing - the time complexity is linear in time for the worst case and for one quantile), then we implement step 4 with a quicksort (with time complexity $\mathcal{O}(n^2)$ in the worst case, but $\mathcal{O}(n \log n)$ for the average case).

We have implemented the two previous views. Our experimental results demonstate that the execution times are very closed but the implementation that is not conform to the original requires less memory: it does not use a merge operation.

Our programs are witten in BSP (bulk synchronous parallel). The BSP model, introduced by Valiant [7] is a general-purpose model in which the properties of the parallel architectures are captured by four parameters: the number of processors p, the time to synchronize the processors, l, the ability of a network to deliver messages under continuous traffic, g, and the speed of the machine. We use BSP both to rely on its prediction capabilities to assist us in the development and its simplicity of programming (SPMD model). We use the standard `BSPlib 1.4` library from Oxford university and PUB7 [8] form the university of Paderborn (Germany).

We have developped codes first for the Direct Remote Memory Access (DRMA) communication facilities implemented with `bsp_put`, `bsp_get`, `bsp_push_register`, `bsp_pop_register` primitives. Some parallel programming libraries require that the data structures used in DRMA operations have to be held at statically allocated memory locations. The BSP toolkits do not have this restriction, which enables communication in certain heterogeneous environments, and allows communication into any type of contiguous data structure including stack or heap allocated data. This is achieved by only allowing a process to manipulate certain registered areas of a remote memory which have been previously made available by the corresponding processes. Second, we have also developped codes for the send/receive paradigm that is also avalaible in BSP (it is more developped with the PUB distribution [9]). The main disadvantage of using DRMA primitives is that the register step can be very time consuming (see addendum), in particular with BSPLib (see [1] for an example). These two programming style allow us to compare the efficiency of communications. DRMA has the advantage that programmer deal with only one primitive to exchange data and there is no notion of 'sender' and 'receiver'.

3.2 Defining our expectations and discussion

Our research is about sorting on *non homogeneous clusters*, we mean clusters with processors at different speeds, interconnection network at different speeds, and also the acces disk time at different speeds. This kind of cluster is of particular interest for those who cannot replace instantaneously with a new generation whole the components of its cluster but shall compose with old and new processors. Note that all the papers (to our knowledge) about parallel sorting algorithms that work (we mean "implemented" on clusters) always consider the special case of homogeneous computing platforms.

Our first expectation is to ensure good load balancing properties. By opposition with the people of the Bekerley NOW project, we argue that the algorithm, based on few parameter that "capture the machine properties", should make the decision. It is not *under* the responsability of some advanced tuning process that we should obtain performances. However we recognize that some technical knowledge (by opposition to the 'algorithmic knowledge') is necessary to avoid paging of virtual memory, to transfer data at peak rates... The algorithm is the priority. In particular, our studies about PSRS like algorithms achieve maximum throughput and very good load balncing properties. Thus, we recognize that the use of automatic configuration tools is necessary prior to lauch the application and these tools should be build by concentrating on the behavior of all the nodes in the system because we would like to deal with non homogeneous clusters). In concentrating on how to obtain an optimal load balancing factor, we cut the discussion about "when to stop communication tuning?". At the user or programmer level, this point is of particular interest.

Our second expectation is that codes we developped should be ported on many systems. As you can see on the addendum, BSP offers many target architectures. The same observation can be done for MPI too but BSP codes have some advantages: it is not necessary to pack/unpack data (the data structures that BSP can capture is not so developped but sufficient in our case), performance tools are available with the Oxford distribution and prediction is allowed.

Third, the question of metrics should be addressed. One of our experiments we have conducted is about the computation of load balancing metrics: (i) the sublist expansion [5] and (ii) the load expansion [4]. The sublist expansion is defined as the ratio of the maximum sublist size to the mean sublist size. The load expansion metric is defined as the ratio of the time spent in sorting sublists on the most heavily loaded processor to the mean time spent over all processors. The sublist expansion metric can be computed with our codes.

Moreover, each code can be tested on eight (integer) benchmarks (we reuse partly those of Bader, Helman and JàJà[4] that we merged in our codes). All the material is available through `http://www.laria.u-picardie.fr/~cerin/=paladin`.

4 Conclusion

In this paper we presented our experience in developping parallel sorting programs based on the PSRS (Parallel Sampling by Regular Sampling) technique and under the framework of

[4]See `http://www.umiacs.umd.edu/research/EXPAR`

BSP.

To our knowledge, little work has been performed about sorting on non homogeneous network. If we try to reuse the main results [10, 3, 2, 4] about sorting on homogeneous clusters as a general strategy, the main difficulty is to show that it will lead to good properties for load balancing, for execution time and also that the underlying algorithm is suitable for a concrete implementation. We are not yet convinced that other technique than the sampling technique will have so much properties. It is still challenging to explore in deep quicksort based approaches and randomized parallel sorting algorithms in the context of non homogeneous clusters.

References

[1] Christophe Cérin and Jean-Luc Gaudiot, "Algorithms for stable sorting to minimize communications in networks of workstations and their implementations in bsp", in *IEEE Computer Society International Wokshop on Cluster Computing (IWCC'99)*, 1999, pp. 112–120.

[2] Hanmao Shi and Jonathan Schaeffer, "Parallel sorting by regular sampling", *Journal of Parallel and Distributed Computing*, vol. 14, no. 4, pp. 361–372, 1992.

[3] X. Li, P. Lu, J. Schaeffer, J. Shillington, P. S. Wong, and H. Shi, "On the versatility of parallel sorting by regular sampling", *Parallel Computing*, vol. 19, pp. 1079–1103, Oct. 1993.

[4] Hui Li and Kenneth C. Sevcik, "Parallel sorting by overpartitioning", in *Proceedings of the 6th Annual Symposium on Parallel Algorithms and Architectures*, New York, NY, USA, June 1994, pp. 46–56, ACM Press.

[5] G. Blelloch, C. Leiserson, and B. Maggs, "A Comparison of Sorting Algorithms for the Connection Machine CM-2", in *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures*, July 1991.

[6] Nicolas Schabanel Afonso Ferreira, "A randomized bsp/cgm algorithm for the maximal independent set problem", *Parallel Processing Letters*, vol. 9, no. 3, pp. 411–422, 1999.

[7] L.G. Valiant, "A bridging model for parallel computation", *Communiations of the ACM*, vol. 33, no. 8, august 1990.

[8] R. Miller and J.L. Reed, "The oxford bsp library : User's guide.", Tech. Rep., Oxford University Computing Laboratory, 1994.

[9] I. von Otte O. Bonorden, B. Juurlink and I. Rieping, "The paderborn university bsp (pub) library - design, implementation and performance", in *13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing, 12 - 16 April, 1999, San Juan, Puerto Rico, available electronically through IEEE Computer Society*, 1999.

[10] M.J. Quinn, "Analysis and benchmarking of two parallel sorting algorithms: Hyperquicksort and quickmerge", *BIT*, vol. 29, no. 2, pp. 239–250, 1989.