PALADIN

[1]

Groupe Paladin au LaRIA

Parallel Sorting: from in−core to out−of−core computation for clusters of workstations

Page developped with Amaya [2] − W3C's Editor/Browser. With Amaya you can edit complex mathematical expressions within HTML pages through a WYSIWYG interface.

Towards Parallel Sorting with Sampling Techniques on non Homogeneous Clusters

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

Abstract:

In this note we introduce some parallel in−core technique for sorting integer keys which is based on the regular sampling technique. We sketch an algorithm which is devoted to clusters of non homogeneous processors (the speeds of processors and/or the speeds to access distributed disks in the clusters are correlated by a multiplicative constant factor and/or the bandwidth of the underlying network interconnected processors by a switch is not to be supposed to be identical in whole the network). This note is devoted to show that sampling techniques are independant (in some way) to the architecture of the computing system on which we run the sorting program. In our case, "independence" means that we can guarantee load

5− Sorting on two sorts of processors with the sampling technique

In this section we exemplify a sorting problem for which we have n data to sort on a system of p=4 processors: two processors are running at 400Mhz and two processors are running at 200Mhz. Moreover assume that the fastest processors are labelled from 1 to 2 and the slower to 3 to 4. Thus, the speed ratio is $k = \frac{400}{200} = 2$.

Let n_1 be the number of data that the processors at 400Mhz should have to deal with to ensure optimal load balancing and let n_2 be the number of data that processors at 200MHz should deal with to ensure optimal load balancing on the 4 processors system. We have:

$$(1)\, n_1 + n_2 = n, \quad (2)\, n_1 - k.n_2 = 0.$$

From the two previous equations, it is not difficult to derive:

$$n_2 = \frac{n}{k+1} \text{ and } n_1 = \frac{k.n}{k+1}$$

In the remainder of the note we impose that $n_1, n_2 \in \aleph$.

To go further in our case study we choose now $n = 72 \Rightarrow n_1 = 48, n_2 = 24$. That is to say that the two fastest processors have 24 data each and the slower processors have 12

BIBLIOGRAPHY:

🟨To view and print the PDF files, the free Adobe Acrobat Reader [7] is required.

See also this web link. [8] or our own bibliography [9].

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 Workshop on Cluster Computing (IWCC'99)*, 1999, pp. 112−120.

2

L.G. Valiant, ''A bridging model for parallel computation'', *Communications of the ACM*, vol. 33, no. 8, august 1990.

3

A. Fabri F. Dehne and A. Rau−Chaplin, ‘‘Scalable parallel computational geometry for coarse grained multicomputers’’, in *Proceeding ACM Symposium on Computational Geometry*, 1993, pp. 298−307.

4

D. Culler R. Karp D. Patterson A. Sahay K.E. Schauser E. Santos R. Subramonian T. von Eicken, ‘‘Logp: Towards a realistic model of parallel computation’’, in *Proceeding 4th ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming*, 1993, pp. 1−12.

5

R. Miller and J.L. Reed, ‘‘The oxford bsp library : User’s guide.’’, Tech. Rep., Oxford University Computing Laboratory, 1994.

6

J. H. Reif and L. G. Valiant, ‘‘A Logarithmic time Sort for Linear Size Networks’’, *Journal of the ACM*, vol. 34, no. 1, pp. 60−76, Jan. 1987.

7

John H. Reif and Leslie G. Valiant, ‘‘A logarithmic time sort for linear size networks’’, in *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, Boston, Massachusetts, 25−27 Apr. 1983, pp. 10−16.

8

Hanmao Shi and Jonathan Schaeffer, ‘‘Parallel sorting by regular sampling’’, *Journal of Parallel and Distributed Computing*, vol. 14, no. 4, pp. 361−372, 1992.

9

balancing in any case.

1−Introduction and motivations

Sorting in parallel records whose keys come from a linearly ordered set has been studied for many years. 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. An important paper dated from 1993 by Guy E. Blelloch, Leonardo Dagum, Stephen J. Smith, Kurt Thearling, Marco Zagha is available on line at http://www3.shore.net/~kht/text/nasa/nasa.htm [12]. Authors propose that sorting be considered an important benchmark for both scientific and commercial applications of supercomputers. As quoted in the previous web link, sorting algorithms can be categorized into three general classes: *counting−based sorts* (counting the number of occurrences of each possible value), *fixed−topology sorts* (use a fixed interconnection network between the processors, such as a hypercube or a grid, and that require no data−dependent communication patterns), and *partitioning sorts* (select a subset of the keys that partition the data and then use these partition elements to route keys to separate sets of processors). There are two main subcategories of partitioning sorts: parallel quicksorts, and sample sorts. Only *sample sorts* is under concern in this note as we will see later.

A sorting algorithm is *stable* if equal keys appear in the output buffer in the same order as in the input buffer (reader should have a look at this page about Shellsort [13] and also Robert Sedgewick pages [14] here). Such tasks typically require that we deal with repeated applications of comparisons and data−movement operations. Even if these operations are performed in parallel, in order to obtain good performance, concrete implementations must take into account not only the algorithmic techniques for merging/partitioning (pipelining, divide and conquer, ranking...) but also the constraints due to the architecture of the machine and the parallel programming language

data each in the initial state. First, each processor sorts the data, then make a selection of pivots.

Let us now considerer the pivots selection problem. The number of selected pivots is $p\,(p-1) = 12$. We pick them at position:

- 6, 12, 18 *i.e.* $\left(\frac{n_1}{p * k}\right)$ and we keep the floor; for the two fastest processors;
- 3, 6, 9 *i.e.* $\left(\frac{n_2}{p * k}\right)$ and we keep the floor; for the two slower processors.

The $p\,(p-1) = 12$ pivots are gathered onto processor 1 (we guess it is a processor at 400Mhz). On this processor, the 12 pivots are sorted. Then we keep $p-1=3$ pivots that define p buckets as follows. It is the part that is a little bit tricky... and we stay on an "informal level" of presentation!

Consider the first pivot in the sorted list. Either there is 3 or 6 elements "at his left" depending on the fact that the pivot was selected from a fast or a slow processor. For each pivot, from 1 from 12 we sum the number of elements the pivot traverses until the sum reaches 24 or 12 depending on the processor we consider (fast or slow). It is clear that the sum does not necessary reach 24 or 12 exactly, so we stop at the pivot position such that the sum does not overpass the desired value (24 or 12).

Then the selected (*we select first pivots for the slower processors*) pivots are distributed to all the processor. After that step we continue with steps 3 and 4 of the PSRS (Parallel Sorting by Regular Sampling) algorithm − see above.

Let us now examine a concrete example. The initial configuration is as follows:

- p1 (400Mhz) contains: 4, 6, 9, 15, 20, 21, 22, 30, 31, 33, 35, 40, 41, 44, 46, 50, 51, 54, 59, 60, 64, 65, 66, 68 and select pivots: 21, 40, 54.
- p2 (400Mhz) contains: 3, 5, 10, 11, 13, 18, 19, 26, 27, 29, 32, 42, 45, 47, 48, 49, 52, 53, 56, 57, 61, 62, 63, 69 and select pivots: 18, 42, 53.

p3 (200Mhz) contains: 2, 7, 12, 14, 23, 24, 25, 34, 37, 58,

X. Li, P. Lu, J. Schaeffer, J. Shillington, P. S. Wong, and H. Shi, ''On the versatility of parallel sorting by regular sampling [10]'', *Parallel Computing*, vol. 19, pp. 1079−1103, Oct. 1993.

10

David R. Helman, Joseph Ja'Ja', and David A. Bader, ''A new deterministic parallel sorting algorithm with an experimental evaluation'', Technical Report CS−TR−3670 and UMIACS−TR−96−54, Institute for Advanced Computer Studies, University of Maryland, College Park, MD, Aug. 1996.

11

Ivan Stojmenovic, ''Constant time bsr solutions to parenthesis matching, tree decoding, and tree reconstruction from its traversals'', *IEEE Transaction of Parallel and Distributed Systems*, vol. 7, no. 2, February 1996.

12

Helman and JaJa, ''Sorting on clusters of SMPs'', *INFRMTCA: Informatica: An International Journal of Computing and Informatics*, vol. 23, 1999.

13

David R. Helman and Joseph Ja'Ja', ''Sorting on clusters of SMPs'', Technical Report CS−TR−3833, University of Maryland, College Park, Nov. 1997.

14

M.J. Quinn, ''Analysis and benchmarking of two parallel sorting algorithms: Hyperquicksort and quickmerge'', *BIT*, pp. 239−250, 1989.

15

used. Despite the great deal of work done in the past on sorting problems using the PRAM (Parallel Random Access Model), it is still challenging to specify algorithms and to implement them efficiently on cluster based systems, we mean systems equipped with fewer commonly used processors interconnected by high speed network with low latency. The special case of *non homogeneous clusters*, we mean clusters with processors at different speeds but from the same manufacturer (a priori) or processors with the same clock speed but with different speeds to access disks and interconnected in the same way (with the same bandwidth), is of particular interest for those who cannot replace instantaneously with a new generation whole the processors of its cluster but shall compose with old and new processors, with old and new disks.

The organization of the note is as follows. In section 2 we review some techniques based on sampling for sorting in parallel. In Section 3 we briefly introduce the BSP model of parallel computation. In section 4 we describe a new parallel algorithm based on *the regular sampling technique* but we deal with the case of non homogeneous cluster in Section 4. We exemplify the case with two kinds of processors. Sections 6 concludes the paper.

2−Related work on sorting

It is well known that the speedup achieved on a parallel distributed machine depends largely on the pertinence of strategies for reducing memory and communication latencies and also on the ways that overhead of scheduling and synchronization are treated. 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 se-

●

67, 71 and select pivots: 12, 24, 37.

● p4 (200Mhz) contains: 1, 8, 16, 17, 28, 36, 38, 39, 43, 55, 70, 72 and selects pivots 16, 36, 43.

(We gave the values after the first initial sorting step which occurs in sequential). The 12 pivots are gathered on processor 1 and sorted. We obtain the following sorted list: 12, 16, 18, 21, 24, 36, 37, 40, 42, 43, 53, 54 The final desired pivots are 18 (because between 12..18 we found 12 elements), 36 (between 21..36 we found 12 elements) and 53 (because between 37 and 53 we found 24 elements). Thus the distribution of data is as follows:

● p1 keeps 40, 41, 44, 46, 50, 51; receives 42, 45, 47, 48, 49, 52, 53 from p2; receives 37 from p3 and receives 38, 39, 43 from p4. Thus p1 produces: 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53. (17 data)
● p2 keeps 56, 57, 61, 62, 63, 69 receives 54, 59, 60, 64, 65, 66, 68 from p1; receives 58, 67, 71 from p3 and receives 55, 70, 72 from p4. p2 produces: 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72. (19 data)
● p3 receives 4, 6, 9, 15 from p1; receives 3, 5, 10, 11, 13, 18 from p2 and receives 1, 8, 16, 17 from p4. It keeps 4, 6, 9, 15. It produces 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18 (18 data)
● p4 receives 20, 21, 22, 30, 31, 33, 35 from p1, receives 19, 26, 27, 29, 32 from p2. It receives 23, 24, 25, 34 from p3. It keeps 28, 36. It produces 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36. (18 data)

Then we conclude that the ordered final sequence is given by the data contained on p3, p4, p1, p2.

Conjecture: under the assumption that $k \leq 2 \wedge (p_1 = p_2)$, in the last step of the algorithm, each processor merges less than $2n_1$ or $2n_2$ data respectively.

Note: we can rearrange at the initial step the processors so that the slower processors are labelled p1 and p2 and the fastest are labelled p3 and p4.

6− Conclusion

In this note we presented some approaches to implement and specify algorithms for (stable) sorting on clusters. The algorithms combine either the good algorithmic and the theoretical properties

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.

16
K. E. Batcher, ''Sorting networks and their applications'', *Proceedings of AFIPS Spring Joint Computer Conference*, pp. 307−314, 1968.

17
David T. Blackston and Abhiram Ranade, ''SnakeSort: A family of simple optimal randomized sorting algorithms'', in *Proceedings of the 1993 International Conference on Parallel Processing. Volume 3: Algorithms and Applications*, P. Bruce Hariri, Salim; Berra, Ed., Syracuse, NY, Aug. 1993, pp. 201−204, CRC Press.

18
A. Borodin and J. E. Hopcroft, ''Routing, merging, and sorting on parallel models of computation'', in *ACM Symposium on Theory of Computing (STOC '82)*, Baltimore, USA, May 1982, pp. 338−344, ACM Press.

19
R. Cole, ''Parallel merge sort'', *SIAM Journal of Computer*, vol. 17, pp. 770−785, Aout 1988.

20
T. Leighton, ''Tight bounds on the complexity of parallel sorting'', in *ACM Symposium on Theory of Computing (STOC '84)*, Baltimore, USA, Apr. 1984, pp. 71−80, ACM Press.

21

lected pivots (2) sort the sublists for which processors are responsible.

Only a MERGE−BASED algorithm is under concern in this paper, principally because the result is easier to obtain: it is closed in spirit to the regular sampling technique (we recall in the next section) for which we have some result in the case of homogeneous clusters. We have experimented in [1 [15]] with such technique for stable algorithms and with the Bulk Synchronous Parallel (BSP) toolkit as the target programming tool. We recall now some information about both about BSP and about sampling techniques that are widely used.

3−BSP: a concrete an efficient programming model

The *bulk synchronous parallel* model, introduced by Valiant [2 [16]] 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 obtain the value of these parameters experimentally. For a given architecture, the parameters measured are obtained by utilizing simple programs: dot product and matrix multiplication for *g*, cost of BSP synchronizations (`bsp_sync()` primitive for *l*, one−to−all and all−to−all for *g*.

In BSP, programmers predict the run length of any computation in terms of these parameters and they can rely on performance tools to automatically perform the task. Thus, one advantage of BSP against competing approaches (CGM [3 [17]], LogP [4 [18]]) is that some libraries (see for instance [5 [19]]) are available for a large number of platforms on which they can effectively run BSP programs. We use the standard `BSPlib 1.4` library from Oxford university.

The BSP model can be regarded as a generalization of the PRAM model where the ideal PRAM is obtained for $l=g=1$ (unitary communication cost and no parallel slackness).

4−Regular sampling: an efficient technique for sorting

In the Bulk Synchronous Programming model (BSP) the sorting problem is approached with a technique known as ''sorting by regular sampling'' [6 [20],7 [21],8 [22],9 [23],10 [24]]. The following link [25] about Parallel Sorting by Regular Sampling (PSRS) is very interesting and provides some codes. Note that the BSP implementation of sorting by the (regular) sampling technique.

To our knowledge, little work has been performed in BSP about the efficiency of load balancing for sorting programs, in particular in the context of non−homogeneous clusters. The methodology exposed in this note is currently under study. We have shown experimentally in [1 [55]] that the algorithms developed for stable sorting in homogeneous clusters are promising. More research involving experiments with input sizes greater than 512kB and 4MB (which are the cache size of our processors) should reveal the impact of cache on performance. The experimental study of the choice of the number of pivots is also important in order to improve performance. Our experiments with the Sujithan implementation (not stable) seem to demonstrate that performance is 2.5 better with p^2 pivots instead of p in all of our implementations in [1 [56]] which are stable.

In conclusion, it should be pointed out that we expect and hope that the results in this note will serve to improve in the future the results of sorting on Network of Workstations in the context of the fastest Disk−to−Disk sort challenge (see `http://now.CS.Berkeley.EDU/NowSort/` [57]) which is stated as follows: ''how much data can you sort in one minute of elapsed time?'' For the 1998 competition, the Berkeley team sorted 8.41 GB in just 1 minute on 95 Ultra1 machines. The reference [27 [58]] about a "Simple randomized mergesort on parallel disks" is also important in the context of out of core computation. The example of Section 5 sketches a solution in the case of two processors which are not identical. This problem is very fundamental

Addendum:

SORTING ON RISC:
If you are interested by sorting problems for RISC processors, please refer to [29 [59], 30 [60], 31 [61]].

OUT−OF−CORE SORTING:
The first reference that we should talk about is the Compact Guide to Sorting and Serching [62] by Thomas Niemann which covers in and out of core sorting. For instance you could get the C−code of the polyphase merge sort of D. Knuth. If you are interested by out of core

D. Nassimi and S. Sahni, ''Parallel permutation and sorting algorithms and a new generalized connection network'', *J. ACM*, , no. 29, pp. 642−667, 1982.

22
C. G. Plaxton, ''Efficient computation on sparse interconnection networks'', Tech. Rep. STAN−CS−89−1283, Department of Computer Science, Stanford University, Sept. 1989.

23
A. Tridgell and R.P. Brent, ''An implementation of a general−purpose parallel sorting algorithm'', Tech. Rep. TR−CS−93−01, Computer Science Laboratory, Australian National University, Feb. 1993.

24
W. D. Frazer and A. C. McKellar. Samplesort: A sampling approach to minimal storage tree sorting. *Journal of the ACM*, 17(3):496−507, 1970.

25
J. H. Reif and L. G. Valiant. A logarithmic time sort for linear size networks. *Journal of the ACM*, 34(1):60−76, January 1987.

26
Y. Won and S. Sahni. A balanced bin sort for hypercube multicomputers. *Journal of Supercomputing*, 2:435−448, 1988.

27
J. S. Huang and Y. C. Chow. Parallel sorting and data partitioning by sampling. In *Proceedings of the IEEE Computer Society's Seventh International Computer Software and Applications Conference*, pages 627−631, November 1983.

by Ronald Sujithan (see `http://www.comlab.ox.ac.uk/oucl/users/ronald.sujithan/` [26]) is not *stable*. The primary goal of the paper referenced [1 [27]] is to introduce BSP implementations for sorting which are stable and based both on the regular sampling technique which provides good experimental results and on a BSR algorithm [11 [28]] for sorting which is stable and stated in a concise way. One of the results obtained in [1 [29]] demonstrate that the use of ShellSort with *k* partitions as the basic stable sequential brick achieves the best performances. Note that in [12 [30],13 [31]], Jájá and Helman say that their algorithm ''can be easily implemented as a stable sort'' and it is devoted to cluster systems, more precisely to multilevel hierarchical memory systems.

The Sujithan implementation describes an efficient implementation of ideas presented in [6 [32],7 [33],8 [34]]. It is based on the following ideas: initially, each processor contains a portion of the list to be sorted. Then a merge−based approach is set up in order to sort the portions and exchange them among all the processors and merged. The merging of the sorted portions in a merge based approach can be achieved either in one step or many steps. References [14 [35],9 [36],8 [37],15 [38]] belong to the category of *one step merge based algorithms*; references [16 [39],17 [40],18 [41],19 [42],20 [43],21 [44],22 [45],23 [46]] belong to *multi steps merge−based algorithms*.

In this note we only consider *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''. A variation of partition based sort is *sample sort* [24 [47], 27 [48], 25 [49], 26 [50]] which is very simple in spirit: (1) a set of *splitters* selected among keys are picked that partition the input into *p buckets* (2) based on their values, the keys are sent to the appropriate bucket − the *i* th bucket goes to the *i* th processor (3) the keys are sorted within each bucket (processor).

It is now well understood that the main practical advantage of *sample sort* is that it greatly reduces the communication required over most of the other sorting algorithms. If there are enough keys per processor (more than *p*) then the splitters can be broadcast to all the processors without a serious overhead, and the data can be routed to its final destination with a single message. However, because of the need to distribute the splitters and the need to sort the sample, it does not work well when there are a small number of keys per processor. The cost of distributing the *p* splitters to each processors can be alleviated by running multiple passes but this adds to the communication costs. Another disadvantage of sample sort is that the buckets are not perfectly balanced at the end. This can require extra memory and extra communication to balance the data.

The Sujithan implementation is as follows (the size of the regular sample is n/p^2):

Step 1:
Perform a local sort; each processor select *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 ≤ i ≤ 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 local sort to merge all the partitions received;

With regard to the time complexity of the above algorithm, it is not difficult to observe that the worst case is dominated by the time cost of step 4 which is ? in the Sujithan implementation because it uses Quicksort. Moreover, some probabilistic arguments (see [6 [51],7 [52],8 [53]]) about pivots arrangements and the redistribution of data can be used to show that the computational cost can match the optimal ? bound.

The other important property of the code is that during step 3, the maximal amount of data communicated by a processor is at most *n*/p thus incurring a BSP communication cost of $g\frac{n}{p}$. In practice, the cost is much smaller.

The implementation of Sujithan is not *stable*: it is based on Quicksort which is not stable and the problem to make Quicksort Stable is still open (see Robert Sedgewick home page at `http://www.cs.princeton.edu/~rs/` [54])). However, it is an ''a priori'' efficient code.

sorting research papers, please refer to [28 [63], 32 [64], 33 [65], 34 [66], 35 [67], 36 [68]]. See also the excellent survey by Jeff Vitter about External Memory Algorithms and Data Structures [69]. An introduction to external Sort can be found here [70]. Please, refer also to the pages about the Dictionary of Algorithms, Data Structures, and Problems [71] and The Stony Brook Algorithm Repository [72].

A PRESENTATION ABOUT SAMPLING:
An introduction to sampling techniques for sorting with many examples which is the presentation (part) that I gave for IWCC'99 and that I use for our master program is available here [73].

CODES:
Find in this section our available codes that implement PSRS and that are developped with PUB7 [74] (university of Paderborn, Germany) and BSPLib [75] (university of Oxford, UK):

- psrs_pub.c [76]: use only `bsp_put` communication primitives and Quicksort as sequential sorting routine.
- psrs_pub_opt.c [77]: use `bsp_sendmsg` communication primitives (message passing routines) and Quicksort as sequential sorting routine.
- psrs_pub_opt_stable.c [78]: use `bsp_sendmsg` communication primitives (message passing routines) and Shellsort as sequential sorting routine. The implementation provides a stable sort.
- psrs_pub_trick_stable.c [79]: use `bsp_sendmsg` communication primitives (message passing routines), QuickSort plus a trick as sequential sorting routine in order to provide a stable sort. The implementation requires approximatively two times more memory storage comparing to the previous codes.
- bench_psrs.tar [80] includes 7 codes (for PUB distribution) with 8 benchmarks (based on those of the university of Mariland [81]): a) `psrs_pub_bench.c` (implementation with DRMA routines), b) `psrs_pub_opt_bench.c` (implementation with send/receive routines) c) `psrs_pub_opt_stable_bench.c` (stable sorting implemented with send/receive) d) `psrs_pub_trick_bench.c`

T

28
R. D. Barve, E. F. Grove, and J. S. Vitter. *Simple randomized mergesort on parallel disks.* In Proc. ACM Symp. on Parallel Algorithms and Architectures, pages 109−118, 1996. To appear in special issue on parallel I/O in ''Parallel Computing'' (see also *Parallel Computing*, 23(4):601−631, 1997).

29
R. Agarwal, *A Super Scalar Sort Algorithm for Risk Processors*, Proc. of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, québec, june 4−6, 1996, pp 240−246

30
Chris Nyberg, Tom Barclay, Zarka Cvetanovic, Jim Gray, Dave Lomet, *AlphaSort: a Risk Machine Sort*, Proc. of the 1994 ACM SIGMOD International Conference on Management of Data, Minneapolis, Minnesota, may 24−27, 1994, pp 233−242

31
Larriba−Pey, J.L.; Jimenez, D.; Navarro, J.J. *An analysis of superscalar sorting algorithms on an R8000 processor*, Proceedings of the 17th International Conference of the Chilean Computer Science Society (SCCC '97), November 12−14, Valpariso, IEEE Computer Society

32
A Framework for Simple Sorting Algorithms on Parallel Disk Systems (extended abstract), SPAA: Annual ACM Symposium on Parallel Algorithms and Architectures, 1998

33
Alok Aggarwal and Jeffrey Scott Vitter, *The Input/Output Complexity of Sorting and Related Problems*, Communications of the ACM, Vol 31, num 9, pp 1116−1127, sep, 1988

34
Jeffrey Scott Vitter and Elizabeth A. M. Shriver, *Algorithms for Parallel Memory {I}: Two−level Memories*, Algorithmica, 1994, aug and sep, vol 12, number 2/3, pp 110−−147

35
Thomas H. Cormen and Melissa Hirschl, *Early experiences in evaluating the parallel disk model with the {ViC}* implementation*, Parallel Computing, vol 23, num 4−5, pp 571−−600, may 23, 1997.

36
Matthew D. Pearson, *Fast Out−of−Core Sorting on Parallel Disk Systems*, Tech Report, 1999, number: PCS−TR99−351, Dept. of Computer Science, Dartmouth College, Hanover, NH, url:

28

R. D. Barve, E. F. Grove, and J. S. Vitter. *Simple randomized mergesort on parallel disks.* In Proc. ACM Symp. on Parallel Algorithms and Architectures, pages 109−118, 1996. To appear in special issue on parallel I/O in ''Parallel Computing'' (see also *Parallel Computing*, 23(4):601−631, 1997).

29

R. Agarwal, *A Super Scalar Sort Algorithm for Risk Processors*, Proc. of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, québec, june 4−6, 1996, pp 240−246

30

Chris Nyberg, Tom Barclay, Zarka Cvetanovic, Jim Gray, Dave Lomet, *AlphaSort: a Risk Machine Sort*, Proc. of the 1994 ACM SIGMOD International Conference on Management of Data, Minneapolis, Minnesota, may 24−27, 1994, pp 233−242

31

Larriba−Pey, J.L.; Jimenez, D.; Navarro, J.J. *An analysis of superscalar sorting algorithms on an R8000 processor*, Proceedings of the 17th International Conference of the Chilean Computer Science Society (SCCC '97), November 12−14, Valpariso, IEEE Computer Society

32

A Framework for Simple Sorting Algorithms on Parallel Disk Systems (extended abstract), SPAA: Annual ACM Symposium on Parallel Algorithms and Architectures, 1998

33

the need to distribute the splitters and the need to sort the sample, it does not work well when there are a small number of keys per processor. The cost of distributing the *p* splitters to each processors can be alleviated by running multiple passes but this adds to the communication costs. Another disadvantage of sample sort is that the buckets are not perfectly balanced at the end. This can require extra memory and extra communication to balance the data.

The Sujithan implementation is as follows (the size of the regular sample is n/p^2):

Step 1:
Perform a local sort; each processor select *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 ≤ i ≤ 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 local sort to merge all the partitions received;

With regard to the time complexity of the above algorithm, it is not difficult to observe that the worst case is dominated by the time cost of step 4 which is ? in the Sujithan implementation because it uses Quicksort. Moreover, some probabilistic arguments (see 6 [51],7 [52],8 [53]]) about pivots arrangements and the redistribution of data can be used to show that the computational cost can match the optimal ? bound.

The other important property of the code is that during step 3, the maximal amount of data communicated by a processor is at most *n*/p thus incurring a BSP communication cost of $g\frac{n}{p}$. In practice, the cost is much smaller.

The implementation of Sujithan is not *stable*: it is based on

(stable sorting for many duplicates i.e. O(n/p) duplicates) e) All the previous codes are implemented with a Quicksort in the first and last steps of the PSRS algorithm. We have also developped codes that use Mergesort (`psrs_pub_merge-sort_bench.c` in the last step and we also developped `psrs_pub_merge_bench.c` that merge sorted chuncks in the last step. Note that merge operations require extra memory (O(n/p)) to do the job. So we have developped a version that do not have the previous drawback: `psrs_pub_quan-tile_bench.c` (implementation based on the search of quantiles in the first step of the PSRS algorithm − 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 search of one quantile is linear in time). The last step of the PSRS is implemented through a Quicksort because quantile search does not produce sorted chunks. It is our most efficient implementation of PSRS. This tar file also includes the shell script `go.sh` in order to do the measures and `test−sort.c` which is a PUB program to evaluate the performance of sequential Quicksort on a single node.

- bspsort.tar.gz [82]: some codes for sorting (stable and not stable) that we developped (historically speaking) for BSPLIB 1.4. The implementation of Sujithan is also included.
- Experimental Parallel Algorithmics [83] at the university of Mariland. This link provides papers and sorting benchmarks both for integers and double precision floating point numbers. Pay attention: the code to generate the benchmarks is available here [84]. See also: EXPAR [85].
- NEW
(sept 2001): all.tar [86] our source codes for out−of−core sorting with a PSRS like algorithm on clusters with processors at different speed. The implementation is done with MPI.

Alok Aggarwal and Jeffrey Scott Vitter, *The Input/Output Complexity of Sorting and Related Problems*, Communications of the ACM, Vol 31, num 9, pp 1116−1127, sep, 1988

34
Jeffrey Scott Vitter and Elizabeth A. M. Shriver, *Algorithms for Parallel Memory {I}: Two−level Memories*, Algorithmica, 1994, aug and sep, vol 12, number 2/3, pp 110−−147

35
Thomas H. Cormen and Melissa Hirschl, *Early experiences in evaluating the parallel disk model with the {ViC}* implementation*, Parallel Computing, vol 23, num 4−5, pp 571−−600, may 23, 1997.

36
Matthew D. Pearson, *Fast Out−of−Core Sorting on Parallel Disk Systems*, Tech Report, 1999, number: PCS−TR99−351, Dept. of Computer Science, Dartmouth College, Hanover, NH, url: ftp://ftp.cs.dartmouth.edu/TR/TR99−351.ps.Z

37
Christophe Cérin and Jean−Luc Gaudiot *Benchmarking Clusters of Workstations Through Parallel Sorting and BSP Librairies, available at this URL [11]*

Quicksort which is not stable and the problem to make Quicksort Stable is still open (see Robert Sedgewick home page at `http://www.cs.princeton.edu/~rs/` [54])). However, it is an ‘‘a priori’’ efficient code.

Link Media Centre

- IEEE Computer Society [87]

With nearly 100,000 members, the IEEE Computer Society is the world's leading organization of computer professionals. Founded in 1946, it is the largest of the 35 societies organized under the umbrella of the Institute of Electrical and Electronics Engineers (IEEE). The Computer Society's vision is to be the leading provider of technical information and services to the world's computing professionals. See also the Task Force on Cluster Computing [88] and the list [89] of the most efficient clusters built today. You must also visit the Distributed Systems Online Channel [90] on Cluster Computing.6.12.99

- The Institute of Electrical and Electronics Engineers (IEEE). [91]

The IEEE ("eye−triple−E"), The Institute of Electrical and Electronics Engineers, Inc., helps advance global prosperity by promoting the engineering process of **creating, developing, integrating, sharing, and applying knowledge** about electrical and information technologies and sciences for the benefit of humanity and the profession. For more on the history of the IEEE see "Our Heritage." [92] The IEEE History Center [93] also has information on the development of electrical and computer engineering and their roles in modern society. 4.12.99

- The Collection of Computer Science Bibliographies [94]

This is a collection of bibliographies of scientific literature in computer science from various sources, covering most aspects of computer science. The about 1200 bibliographies are updated monthly from their original locations such that you'll always find the most recent versions here. The collection currently contains more than one million references (mostly to journal articles, conference papers and technical reports) and consists of 660 MBytes of BibTeX entries. More than 9000 references contain crossreferences to citing or cited publications. More than 80,000 references contain URLs to an online version of the paper. There are more than 2000 links to other sites carrying bibliographic information. 1.12.99

- ResearchIndex [95]

ResearchIndex is a scientific literature digital library that aims to improve the dissemination and feedback of scientific literature, and to provide improvements in functionality, usability, availability, cost, comprehensiveness, efficiency, and timeliness. Rather than creating just another digital library, ResearchIndex provides algorithms, techniques, and software that can be used in other digital libraries. ResearchIndex indexes Postscript and PDF research articles on the Web. 30.1.01

- IEEE Xplore [96]

Introducing IEEE Xplore , the dynamic new online delivery system for IEEE technical and scientific information. You can now conduct your research from a single, desktop platform. Browse the complete collection of tables of contents of IEEE. transactions, journals, magazines, conference proceedings, and standards. Search and view all IEEE abstract/citation records starting from 1988 (expansion of service previously provided through Bibliographies Online). Browse, search and view full−text articles of your personal online subscriptions (previously provided through OPeRA, the Online Periodicals and Research Area), and "IEEE Spectrum" magazine (available to IEEE members only). For additional information refer to the Frequently Asked Questions [link to http://ieeexplore.ieee.org/lpdocs/epic03/faq.htm [97] on the IEEE Xplore home page. 1.12.99

- Computer Science Bibliography (dblp.uni−trier.de) [98]
2.10.99

- Collection of Lecture Notes, Surveys, and Papers [99]

This is a collection of some lecture notes, papers, ... that we have found in Internet. By no way this is a complete archieve, its primary aim is to serve the local needs of our research group and our students. So many of items are accessible only from Paderborn. −− Artur Czumaj (artur@uni−paderborn.de) −− Miroslaw Kutylowski (mirekk@uni−paderborn.de) 2.10.99

- THE WWW VIRTUAL LIBRARY at INRIA (FRANCE) [100]

2.10.99

- BSP (Bulk Synchronous Parallel model) Worldwide [101]

BSP Worldwide is an association of people interested in the development of the Bulk Synchronous Parallel (BSP) computing model for parallel programming. It exists to provide a convenient means for the co−ordination of all kinds of work on BSP including: Research into properties of the model, Application of the model to programming tasks of all kinds including the scheduling, of parallel execution, Performance benchmarking and comparison with other approaches, Cost modelling and performance prediction, Definition of standard functions for programming in the BSP style and for the standardisation of these functions, Implemenation of programming tools to support the use of the model. The organisation has yet to formalise its structure. 2.10.99

- BSP Headquaters [102]

This page contains links to my tutorials which demonstrate how to use BSP. Most of these tutorials were originally written for Quake, but I have added Hexen2 and Quake2 versions, so you can use them to familiarize yourself with the editor while working on a game you enjoy. So, pick a lesson from the list and get started. Note, a zip file of each lesson is provided if you would prefer to download it and read it offline. 2.10.99

- RESSOURCES INTERNET, Mathématiques, Informatique [103]

This site is located at the university of Jussieu (Paris, France). It is developped for the university library and you will find many resources. The page is in french only. 2.10.99

- NCSTRL: This server operates at UW Madison Computer Sciences Technical Reports. Fielded search of the collection [104] can be done on this server. 2.10.99

- Supercomputing Centers and Parallel Computing [105]

This site is located at the university of Mannheim (Germany). See also the Linux Parallel Computing at the University of Mannheim here [106] and also Linux Parallel Processing Using SMP here [107]. However, these two last links seem not to be up to date. 2.10.99

- The sites listed below are about RAID architecture : Raid Advisory board [108]

For a definition see: RAID [109] Reffer to the article entitled RAID Performance Evaluation [110] and the article entitled A Performance Evaluation of RAID Architectures [111] by Shenze Chen and Don Towsley. The Berkeley IS-TORE [112] Project is building adaptive, highly−available, self−maintaining, self−tuning back−end servers for storage−intensive network services, and is investigating the programming abstractions for defining their adaptive behavior. At Berkeley, see also the Tertiary Disk Project [113]. 2.10.99

- Virtual Librarian at Monash University [114]

This site has been written by Monash University librarians to teach the skills and tools needed to find information. More tutorials and subject guides will be added in the near future. Your comments on the pages currently available are appreciated by the library, and can be sent via email, or by filling in a short survey. 2.10.99

**Towards Parallel Sorting
with Sampling Techniques on
non Homogeneous Clusters**

[1] *href*=http://www.laria.u−picardie.fr/PALADIN/ 1
[2] Amaya *href*=http://www.w3.org/Amaya 1
[3] LaRIA On−Line Homepage *href*=http://www.laria.u−picardie.fr 1
[4] News *href*= 1
[5] Partners *href*= 1
[6] French *href*= 1
[7] Acrobat Reader
href=http://www.adobe.com/prodindex/acrobat/readstep.html#reader 1
[8] web link. *href*=http://now.cs.berkeley.edu/NowSort/sort.long−bib.html. 1
[9] bibliography *href*=http://www.laria.u−picardie.fr/~cerin/=paladin/these.bib . . . 1
[10] On the versatility of parallel sorting by regular sampling
href=http://www.cs.utoronto.ca/~paullu/Papers/psrs.ps.Z 3
[11] URL *href*=ftp://www.laria.u−picardie.fr/~cerin/=paladin/ppl.ps.gz. 7
[12] http://www3.shore.net/~kht/text/nasa/nasa.htm
href=http://www3.shore.net/~kht/text/nasa/nasa.htm 7
[13] Shellsort *href*=http://www.bell−labs.com/user/suel/papers/papers.html. 7
[14] pages *href*=http://www.cs.princeton.edu/~rs/cs226/lectures/lectures.txt. . . . 7
[15] 1 *href*=paladin−sorting.html#CERIN99 7
[16] 2 *href*=paladin−sorting.html#valiant 7
[17] 3 *href*=paladin−sorting.html#Dehne 7
[18] 4 *href*=paladin−sorting.html#culler93 7
[19] 5 *href*=paladin−sorting.html#MR94 7
[20] 6 *href*=paladin−sorting.html#reif87 7
[21] 7 *href*=paladin−sorting.html#STOC::ReifV1983 7
[22] 8 *href*=paladin−sorting.html#ShiHanmaoa1992a 7
[23] 9 *href*=paladin−sorting.html#LiXandLuPa1993a 7
[24] 10 *href*=paladin−sorting.html#HelmanDavi1996b. 7
[25] link *href*=http://sdcd.gsfc.nasa.gov/ESS/eazydir/inhouse/mobarry/sorting/ . . . 7
[26] `http://www.comlab.ox.ac.uk/oucl/users/ronald.sujithan/`
href=http://www.comlab.ox.ac.uk/oucl/users/ronald.sujithan/ 7
[27] 1 *href*=paladin−sorting.html#CERIN99 7
[28] 11 *href*=paladin−sorting.html#S96 7
[29] 1 *href*=paladin−sorting.html#CERIN99 7
[30] 12 *href*=paladin−sorting.html#HelJaJ99 7
[31] 13 *href*=paladin−sorting.html#UMCP−CSD__CS−TR−3833. 7
[32] 6 *href*=paladin−sorting.html#reif87 7
[33] 7 *href*=paladin−sorting.html#STOC::ReifV1983 7
[34] 8 *href*=paladin−sorting.html#ShiHanmaoa1992a 7
[35] 14 *href*=paladin−sorting.html#Quinn. 7
[36] 9 *href*=paladin−sorting.html#LiXandLuPa1993a 7
[37] 8 *href*=paladin−sorting.html#ShiHanmaoa1992a 7
[38] 15 *href*=paladin−sorting.html#spaa94_46 7