Sparse computations and Multi-BSP

A. N. Yzelman

May 9, 2016

BSP is a simple but effective model for parallel computing. A BSP computer (p, g, l) consists of p sequential processors with sufficient local memory. A black-box network interconnects these. This network is assumed to be full duplex and optimised for all-to-all communication; a processor can thus simultaneously send and receive a single data word at constant cost g. The network has an associated latency cost l that has a more complete interpretation in the following context.

A BSP algorithm $(p, w_{i,s}, h_i)$ consists of a series of supersteps. It is described as a sequential program parametrised in p as well as in the process ID $s \in \{0, 1, \ldots, p-1\}$. A superstep consists of computation and communication. Computation is a set of operations on local data using only local processor resources. Communication is a global action that involves only the network and is disallowed from affecting computation that occur in the same superstep, and vice versa. Computation at the ith superstep of process s may assume that all communication in preceding supersteps, as far as they affect the state of memory local to process s, have completed. A program needs the network to determine whether it is safe to advance a superstep; this is exactly the latency cost l introduced earlier.

The sequential work in the ith superstep at process s has cost $w_{i,s}$. The number of data words sent and received by process s as part of the ith superstep are $t_{i,s}$ and $r_{i,s}$, respectively. The *h-relation* of the ith superstep, the cost of the process that forms the *communication bottleneck*, is $h_i = \max_s \max\{t_{i,s}, r_{i,s}\}$: the cost incurred by the process which has to send or receive the largest number of data words of all processes during superstep i.

Given a BSP computer (p, g, l) and a BSP algorithm $(p, w_{i,s}, h_i)$, the cost of running that algorithm on that computer is

$$T = \sum_{i=0}^{N-1} \max\{\max_s w_{i,s} + l, h_i g + l\}.$$

This is the original BSP cost model by Valiant [2].

BSP fails to penalise non-local data movement. For instance, when one core initiates data movement to another core which share an L2 cache, this will be more efficient than movement which instead needs to pass through some higher-level resource. The same holds true for data movement between sockets and nodes; architectures are becoming increasingly non-uniform in memory access.

Multi-BSP, developed by Valiant in the late 2000s [3], alleviates this problem by a recursive computer model definition: a Multi-BSP computer consists of p sequential computers, *or* p sub Multi-BSP computers. This leads to a tree of interconnects with depth d, where every node furthermore is equipped with local memory.

A Multi-BSP computer is characterised by four parameters at each of the d levels of the tree: (p_i, g_i, l_i, M_i) describes the Multi-BSP machine at level i. It has 1) p_i sub Multi-BSP machines, 2) a full-duplex all-to-all network with BSP characteristics g_i and l_i connecting the submachines, and 3) a local data storage with a capacity of M_i data words. The capacity of the root machine itself is assumed sufficient to store the problem at hand.

A good Multi-BSP algorithm minimises the data movement through the various levels of the Multi-BSP computer, while 1) penalising far-away data movement and 2) properly modelling the data flow from main memory down to processing cores through the complete memory hierarchy. The remainder of this abstract discusses the implications of this model on various sparse matrix computing problems. While explicit Multi-BSP programming is possible in MulticoreBSP [5], the presented insights are not restricted to it and should lead to practical algorithms independent of the chosen implementation framework.

Consider an $m \times n$ sparse matrix A which contains nz nonzeroes $a_{ij} \in \mathcal{V}$. Hypergraph modelling enables efficient parallelisation of operations on A: in the most generic form, the nonzeroes $\mathcal{V}$ correspond to vertices, while rows $n_i^{\text{row}} \subseteq \mathcal{V}$ and columns $n_j^{\text{col}} \subseteq \mathcal{V}$ of A correspond to the hyperedges $\mathcal{N}$. A distribution of work corresponds to a partitioning of $\mathcal{V}$, leading to process-local submatrices A_s. Other hypergraph models may collapse information to increase partitioning efficiency; Bisseling et al. [1] present an overview and comparison.

For the sparse matrix–vector (SpMV) multiplication $Ax = y$ with x, y of appropriate size, a scalable parallel algorithm proceeds in three stages: 1) gather non-local elements required for the multiplication with A_s, 2) perform the local multiplication with A_s, and 3) collect remote contributions to local output vector elements. The hyperedges encode information on the communication cost via their connectivity λ; the volume of data movement is captured exactly by the $\lambda - 1$ metric on $\mathcal{N}$ [1, 4]. Contemporary sparse matrix partitioners minimise this volume under the load-balance constraint $\max_i |\mathcal{V}_i| \leq (1 + \epsilon)nz/p$. The actual h-relation incurred in steps 1 and 3 are minimised by finding appropriate vector distributions of x, y, *given* a distribution for A. This highly BSP-centric approach minimises and balances both computation and communication.

In a Multi-BSP approach, a P-partitioning of $\mathcal{V}$ should be tuned so that for each part $\mathcal{V}_i$, the corresponding local computation $y_s = A_s x_s$ requires less than M_0 data words to store. Note that $P \gg p$ for large enough matrices, thus, seemingly(!), leading to a very different approach. All elements from A, x, y must be streamed from the root down to the leaves at least once, thus incurring a minimum cost of $(nz + m + n)\Pi_{i=0}^{d-1} g_i$. Vector elements, however, are re-used and the volume of data movement can be minimised by recursively partitioning at each level of the Multi-BSP tree. This differs from recursive bipartitioning in two ways: 1) the notion of load balance includes the number of non-empty rows and columns, apart from the number of nonzeroes; and 2) at each level, the load-balancing constraint is reset to target construct parts of M_i data words each.

Now, each level i of the Multi-BSP tree runs its own SPMD program which 1) retrieves from our parent the next block y_s, A_s, x_s that together take at most M_i data words; 2a) if $i > 0$, recursively call this program on every submachine, or 2b) perform a local SpMV multiplication $y_s = A_s x_s$; 3) accumulate remote contributions to elements of y_s owned by us; 4) write back to the parent the part of y_s owned by us; and 5) if this A_s was the last block, yield to the parent program. Note that this is remarkably close to the original BSP algorithm. This algorithm extends to symmetric matrices by simple adaptation of step 2b.

Via a frame of thought prescribed by the Multi-BSP framework, we thus refined and extended the cache-oblivious algorithm by Yzelman and Bisseling [4]. We apply the same ideas to the unstructured sparse matrix powers (SpMP) kernel, and additionally allows present and quantify trade-offs for 2.5D counterparts to all aforementioned kernels.

References

[1] R. H. Bisseling, B. O. Fagginger Auer, A. N. Yzelman, T. van Leeuwen, and Ü. V. Çatalyürek. Two-dimensional approaches to sparse matrix partitioning. In U. Naumann and O. Schenk, editors, *Combinatorial Scientific Computing*. CRC Press, 2012.

[2] L. G. Valiant. A bridging model for parallel computation. *CACM*, 33(8), 1990.

[3] L. G. Valiant. A bridging model for multi-core computing. *JCSS*, 77(1), 2011.

[4] A. N. Yzelman and R. H. Bisseling. Two-dimensional cache-oblivious sparse matrix–vector multiplication. *ParCo*, 37(12), 2011.

[5] A. N. Yzelman, R. H. Bisseling, D. Roose, and K. Meerbergen. MulticoreBSP for C: a high-performance library for shared-memory parallel programming. *IJPP*, 42, 2014.