Multi-DaC Programming Model

A variant of Multi-BSP Model for divide-and-conquer Algorithms

Abdorreza Savadi
Ferdowsi University of Mashhad (FUM), Mashhad, Iran
savadi@um.ac.ir

Morteza Moradi Hossein Deldari
Islamic Azad University-Mashhad Branch (IAUM),
Mashhad, Iran
moradi.edu@gmail.com/hd@um.ac.ir

Abstract

Nowadays, the evolution of multi-core architectures goes towards increasing the number of cores and levels of cache. Meanwhile, current typical parallel programming models are unable to exploit the potential of these processors efficiently. In order to achieve desired performance on these hardwares we need to understand architectural parameters appropriately and also apply them in algorithm design. Computational models such as Multi-BSP, illustrate these parameters and explain adequate methods for designing algorithms on multi-cores. One of the most applicable categories of problems is Divide-and-Conquer (DaC) that needs to be adapted by such model for implementing on these systems.

In this paper, we have attempted to make a mapping between DaC tree and the Memory Hierarchy (MH) of multi-core processor. Multi-BSP model inspired us to introduce Multi-DaC programming model. Analogous to Multi-BSP analysis, lower bounds for communication and synchronization costs have been presented in the paper respecting DaC algorithms. This work is a step towards making multi-core programming easy and tries to obtain correct analysis of DaC algorithm behavior on multi-core architectures.

Categories and Subject Descriptors D.1.3 [*Concurrent Programming*]: Parallel Programming; D.3.2 [*Language Classifications*]: Concurrent, Distributed and Parallel Languages; D.4.8 [*Performance*]: Modeling and prediction

General Terms Multi-BSP, parallel, algorithms, Divide-and-Conquer, multi-core architectures, Skeletal Programming, cache, Memory Hierarchy

Keywords Multi-BSP, parallel, Divide-and-Conquer, multi-core architectures, cache, Memory Hierarchy

1. Introduction

The multi-core architectures were introduced as a remedy to avoid the limitations which were made during development of parallel architectures such as high energy consumption, heat dissipation, and the memory-wall problem. Traditional programming models don't consider existence of shared memory hierarchy on a processor therefore using these models will produce undesired results. In void of proper multi-core software development models, designing efficient programs have become awkward, so there is a demand for revising current models or introducing new ones.

On the other hand, it is predicted that in the forthcoming years the number of cores on a chip will increase in large numbers and MH will get larger in more levels respectively. Hence, if these architectures want to stabilize their place among parallel systems have to be accompanied by appropriate programming tools and models because the complexity of efficient algorithm design for them will get higher soon.

In order to simplify programming, it seems we should distinguish efficient system parameters to be able to add a layer over architecture to hide its complexity from programmers. Programming models can well show efficient performance parameters and are able to lessen the difficulty of algorithm design by explaining the style of programming. So, designing a robust flexible model that can fit to all needs and future architectures is a high goal.

In this paper, we attempted to step forward towards achieving the mentioned goal by presenting a programming model for Divide-and-Conquer (DaC) algorithms which are widely applied species of algorithms. The proposed model is based on adapting DaC to a computational model for multi-cores called Multi-BSP. The benefit of our model is that it can anticipate the cost of algorithm execution on hardwares which have a hierarchical shared memory.

In the following section the related works are presented. In the Section 3 we introduce the basic description of our model. In Section 4 we describe mapping algorithm and in Section 5 scheduling algorithm used in the model is presented. Lower bounds for estimating the costs are demonstrated in Section 6. Two case studies, merge sort and convex hull, are evaluated in Section 7. The conclusions and future works appear in the last section.

2. Related Works

With the advent of multi-cores, the need to exploit their potential was appeared. One of the efforts undertaken for fulfilling the above need is modeling the hierarchical memory parallel architectures to make the ability of their performance analysis. Computational models for parallel and distributed systems such as PRAM[8], LogP[6], QSM[13], and BSP[17] can be the apt candidate for the purpose, but as they have not included the hierarchical memory on the scale of a single chip, they do not have the required efficacy for these architectures. Recent works have been done on hierarchical single-core processors. For example, PEM is a two level multi-processor model in which, the first level is related to private cache while the second level is an external shared memory among processors. The model is based on two previous models of Ideal Cache Model[1] and Two Level I/O Model[9]. Multi-core Cache Model[5] is more extended than PEM. In stride of evolution of multi-core architectures, it is determined that with the increase in

DAMP'12, January 28, 2012, Philadelphia, PA, USA.
Copyright © 2012 ACM 978-1-4503-1117-5/12/01...$10.00

core numbers and growth in memory bus contention and existence of memory latency, the levels of cache must be increased to compensate these problems. Therefore, some architectures came out with more cache levels, and new computational models appeared respectively[4][3]. In the past, UMH[2] and MHG[15] were considered as multi-level cache models, but they are not suitable for multi-cores. Recently, a model that is called Universal multi-core Model[16] has extended MHG and uses it in multi-cores. In this regard, in[5], HM model has enhanced Multicore Cache model. The most distinguished model for multi-cores is discussed by Valiant which is called Multi-BSP[18]. In this model, the architecture is described by a set of parameters corresponding to size, latency, and levels of memory and it attempts to map supersteps in architecture in order to make a bridge between multi-core architecture and BSP programming.

According to Valiant's opinion[18], Multi-BSP is a proper choice for recursive algorithms, and since DaC has a recursive structure, it is a good match for Multi-BSP adaptation. Although Multi-BSP conforms to recursive algorithms, just specific recursive ones can be implemented by this model, therefore, this fact can restrict the application of this model in parallel computing. Our work removes this restriction from DaC algorithms.

3. Multi-DaC Programming Model

First, following three definitions are presented:

Definition 3.1. *Task Dependency Tree (TT), is a result of DaC algorithm execution so that leaf nodes execute sequntial algorithm on input data, and non-leaf nodes merge the output of their children. In this tree the branching factor is K that is an integer number.*

Definition 3.2. *Memory Hierarchy Tree (MT), represents the hierarchical structure of multi-core architecture. In this tree, leaf nodes (lowest level) and non-leaf nodes are mapped on the processor cores and cache/memory components respectively.*

Definition 3.3. *The required memory for executing a sequential algorithm with input size of t is $m_s(t)$; this memory includes input, output, and temporary space needed during the execution. Also, $m_m(t)$ is the required memory for a merge algorithm execution.*

Presumptions in Multi-DaC model are as follows:

1. Architectural detail description used in this model are based on definitions of Multi-BSP model[18]. *"An instance of a Multi-BSP is a tree structure of nested components where the lowest level or leaf components are processors and every other level contains storage capacity"*[18]. A level i component of MT, $i = 1..d$ and d is depth of MT levels, is shown by (p_i, g_i, L_i, m_i) where p_i is the number of level $i-1$ components inside a level i component. g_i is the communication bandwidth parameter, which is the ratio of the number of operations that a core processor can perform in a second to the number of words that can be transmitted in a second between the memories of a component at level i and its parent component at level $i+1$. And, L_i is the cost charged for barrier synchronization of a level i superstep. Also, m_i is the number of words of memory inside a level i component that is not inside any level $i-1$ component. Moreover, the last level of memory(root) is assumed to be unlimited, and in the lowest level, $g_0 = 1$ and $L_1 = 0$.
2. The DaC is considered to be regular. The task dependency tree(TT) of a regular DaC is a complete K-tree and its depth depends on the size of input. For example merge sort, matrix multiplication and FFT all are regular but quick sort is not.
3. Cache memory hierarchy runs in inclusive mode where any word at one level has a distinct copy at every higher level.

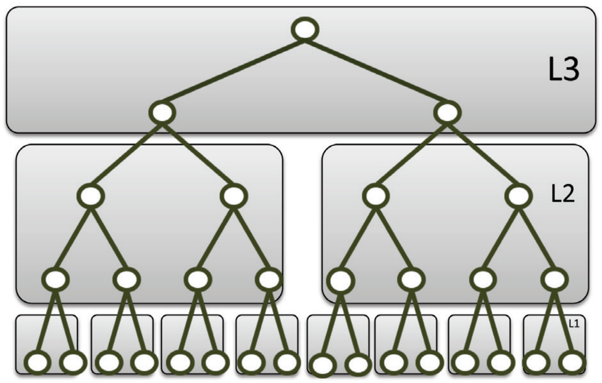

Figure 1. An example for Mapping TT to MT

4. Mapping To Memory Hierarchy

Multi-DaC model proposes a mapping from TT to MT in a way that each node of TT is logically placed on a level of MT that has equal or larger memory size than needed memory for the node execution. (See Fig. 1.) The required steps for mapping from TT to MT includes:

1. **Level Mapping**: Maps one level of TT to one level of MT.
2. **Component Mapping**: Maps TT nodes which are mapped to one level of MT, to components of that level.
3. **Core Mapping**: Maps nodes which are mapped to a component, to cores of that component.

4.1 Level Mapping

In the usual cases, the input size of problem (n) is huge, so the number of TT levels (h) would be higher than MT levels. According to pigeonhole principle, some MT levels receive $\lceil \frac{h}{d} \rceil$ or more of TT levels. Logically, each level of TT must be mapped to a level of MT that has memory size equal or greater to the needed memory by the TT level nodes.

Lemma 4.1. *Suppose n is the input size, K is the branching factor, and h is the depth of TT. Then size of needed memory for a level i node of TT is:*

$$V_i = \left(\sum_{j=i}^{h-1} K^{j-i}\, m_m(\frac{n}{K^j}) \right) + K^{h-i}\, m_s(\frac{n}{K^h}) \qquad (1)$$

Proof. Since in a DaC tree, sequential and merge algorithm are carried out in leaf and non-leaf nodes respectively, the following recursive formula can compute the needed memory size that a level i node in TT requires.

$$V_i = \begin{cases} m_s(\frac{n}{K^h}) & i = h \\ K\, V_{i+1} + m_m(\frac{n}{K^i}) & 0 \le i < h \end{cases} \qquad (2)$$

Each TT node is either leaf or root subtree. In Formula 2. the first part corresponds to required space for sequential execution of a leaf. The second part is the needed space for receiving K children's output and merging them in the node(children's parent). Regarding to the inclusive behavior of memory levels, Formula 2 shows the overall required memory for a level i node of TT. Then the iteration form of V_i will be:

$$V_i = m_m(\frac{n}{K^i}) + K m_m(\frac{n}{K^{i+1}}) + K^2 m_m(\frac{n}{K^{i+2}}) + \cdots +$$

$$+ K^{h-i-1} m_m(\frac{n}{K^{h-1}}) + K^{h-i} m_s(\frac{n}{K^h})$$

Therefore, the Formula 1 will be trivial. □

Definition 4.2. *(Level Mapping) The mapping of a level i node of TT to a level of MT is as follows:*

$$r(i) = \{\min(j) | m_j \geq V_i\}$$

where i is the level number of the node in TT and j is the number level in MT. And also V_i is the sum of needed memory for its input, output, and children.

Definition 4.3. *The level ith of MT receives levels of TT that are between $fst(i)$ and $lst(i)$ where :*

$$fst(i) = \{\min(j) | r(j) = i\}$$

$$lst(i) = \{\max(j) | r(j) = i\}.$$

4.2 Component Mapping

Each node of TT that is mapped to a level of MT must lie on a component in that level. In this mapping, for each component, a queue will be formed. Then, the levels of TT that is mapped to the component will be assigned to the queue in bottom-up manner i.e. lower level nodes of TT receive higher ranks in the queue.

Lemma 4.4. *The length of queue for mapped nodes of ith level of TT on a component of MT, where K is the branching factor of TT, is as follows:*

$$S_i = \begin{cases} 1 & i = 0 \\ \left\lceil \frac{S_{i-1}K}{p_{r(i)}} \right\rceil & i = fst(r(i)) \\ S_{i-1}K & \end{cases} \quad (3)$$

Proof. In the first level of TT , we have just one node therefore the queue length for that level would be one.

The times of mapping nodes of ith level of TT to a component of MT is multiplier of their parent's queue length where $r(i)$ is corresponding level of MT and $p_{r(i)}$ is the number of subcomponents so we have $\left\lceil \frac{S_{i-1}K}{p_{r(i)}} \right\rceil$.

In the case that nodes are mapped into the component of their parent i.e. $i \neq fst(r(i))$ must be executed by the component and the queue length for them would be $S_{i-1}K$.

□

In a level superstep[18], all computations for the ingress node must be finished in order to complete the superstep. Hence, the number of superstep for a level i component is $S_{fst(i)}$.

4.3 Core Mapping

According to Multi-BSP[18], the number of cores for a level j component of MT is equal to:

$$P_j = p_1 p_2 ... p_j.$$

In this step, the dequeued node must be affined to a core. This affinity can be expressed by:

$$C = f + i$$

where f is the first core number in parent's component, and i is the corresponding core number that is determined by round-robin algorithm when $0 \leq i < P_j$. Finally, C is the assigned core number.

5. Relationship with Multi-BSP

In this section the scheduling of Multi-BSP model is reviewed and the mapping required for conforming to Multi-DaC, that is a variant of Multi-BSP model, is presented.

In Multi-BSP, program execution commences in a component of highest memory level and continues to the last level. In each component input data must be divided into equal parts that are less or equal than subcomponents' memory size. Because the number of these parts is likely to be more than the number of destination subcomponents, sending them must be done in phases. Each phase is *a level i superstep that is a construct within a level i component that allows each of its p_i level $i-1$ components to execute independently until they reach a barrier. When all p_i of these level i - 1 components have reached the barrier,they can all exchange information with the m_i memory of the level i component with communication cost as determined by g_{i-1}*[18]. Designed algorithms for Multi-BSP must have desired variant branching factor corresponding to each memory level. Finding such algorithms seems laborious.

To alleviate the complexity of designing Multi-BSP algorithms, Multi-DaC model tries to present a mapping between divide-and-conquer algorithms and Multi-BSP without need to large changes in conventional DaC algorithms. In this model in contrast to Multi-BSP, divisions are based on a constant factor, task dependency branching factor. To reach the desired part size, TT must be expanded within the component until the expanded levels can be sent to subcomponents.

Multi-DaC superstep contains the execution of component's ingress node of TT and all its subtree. Superstep starts with sending a node to a component, continues with creating the node subtree, and ends with merging the levels of the subtree. Analogous with Multi-BSP, a superstep in Multi-DaC, includes the supersteps of its component's subcomponents that are synchronized by barrier mechanism and share their results on the parent component.

To avoid loss of productivity in performing merge operations, Multi-DaC model benefits from running multiple merge operations concurrently on the subset of cores that are correspond to the component.

6. Lower Bounds

According to the proposed mapping and the scheduling algorithm, conforming to the lower bounds presented in [18], the total communication and synchronization cost of an algorithm on Multi-DaC model can be computed as follows:

Theorem 6.1. *The cost of communication caused by a divide-and-conquer algorithm with branching factor of K and input size of n on a multi-core architecture defined by Multi-BSP model with depth of d is at least:*

$$comm(n, d) \geq \sum_{i=1}^{d-1} S_j V_j g_i$$

when j is $fst(i)$.

Proof. The overall communication caused by a level i superstep, according to descriptions presented in Section 5 is equal to the memory consumption by the first level node of TT mapped to the ith level of MT. So, the total communication size for the ith level superstep using Lemma 4.1 and Defintion 4.3 is $V_{fst(i)}$. Also, the number of supersteps for the level ith component using Lemma 4.4 is $S_{fst(i)}$. Therefore, total communication cost for level ith component is $S_j V_j g_i$. Consequently, the overall communication cost caused by the algorithm can be computed by summarizing cost of all memory levels. □

Theorem 6.2. *The cost of synchronization caused by a divide-and-conquer algorithm with input size of* n *on a multi-core architecture defined by Multi-BSP model with depth of* d *is:*

$$synch(n,d) \geq \sum_{i=1}^{d} \sum_{j=fst(i)}^{lst(i)} S_j \; L_i$$

Proof. Using Definition 4.3, Lemma 4.4 and descriptions presented in Section 5, each mapped TT level where j is the level number TT must be synchronized S_j times in order to get processed. Therefore, the total cost for a level i is:

$$\sum_{j=fst(i)}^{lst(i)} S_j \; L_i$$

and for all of levels of MT results the theorem. □

7. Case Studies

In this section, two parallel algorithms, Merge Sort and Convex Hull, are analyzed and simulated based on Multi-DaC model.

7.1 Merge Sort Algorithm

Here, we used the algorithm presented by[11]. Regarding to the model and the fact that MergeSort halves data into two equal sections, the value of K will be 2. Also, the memory complexity for sequential portion and merge stage are as following:

$$m_s(n) = O(n\log_2 n)$$

$$m_m(n) = O(n)$$

7.2 Convex Hull Algorithm

There are some Convex Hull algorithms such as quickHull[7], Graham scan[10] and divide-and-conquer[12]. The algorithm that is used here is the divide-and-conquer algorithm presented for two dimensional points. Similar to MergeSort, the algorithm[12] tries to partition points into two equal sections, so the value of K will be 2.

The memory complexity for sequential and merge stages are as follows:

$$m_s(n) = O(n\log_2 n)$$

$$m_m(n) = O(n)$$

7.3 Simulation

The simulations presented in the paper, are performed by SuperESCalar Simulator(SESC) that is introduced in[14]. SESC is a cycle-accurate architecture simulator which allows defining a desired multi-core with complex memory hierarchy. In order to evaluate the model, two experiments have been designed based on increasing the number of cores. For the experiments, Table I shows the configurations used in the experiments.

The overall execution time of an algorithm in this model is calculated by:

$$Exec(n,d) + Comm(n,d) + Synch(n,d)$$

- Merge Sort: As time complexity for merge stage is $O(n)$ and for sequential execution is $O(nlogn)$, then parallel time is

$$Exec(n,d) = \sum_{i=0}^{h-1} 2(\frac{n}{K^i})S_i + (\frac{n}{K^h}\log\frac{n}{K^h})S_h$$

  Figure 2 shows a comparison between simulation results and outcome of the analytical model for estimated execution time.

Table 1. Used MT-configurations in the experiments

Cores	Configurations($(\mathrm{p_i}, \mathrm{g_i}, \mathrm{L_i}, \mathrm{m_i})$, i=1..4)
4	(1,1,0,32KB) (2,3,$\mathrm{L_2}$,128KB) (2,38,$\mathrm{L_3}$,512KB) (1,∞,$\mathrm{L_4}$,∞)
8	(1,1,0,32KB) (4,3,$\mathrm{L_2}$,256KB) (2,38,$\mathrm{L_3}$,1MB) (1,∞,$\mathrm{L_4}$,∞)
16	(1,1,0,32KB) (4,3,$\mathrm{L_2}$,256KB) (4,38,$\mathrm{L_3}$,2MB) (1,∞,$\mathrm{L_4}$,∞)
32	(1,1,0,32KB) (4,3,$\mathrm{L_2}$,256KB) (8,38,$\mathrm{L_3}$,4MB) (1,∞,$\mathrm{L_4}$,∞)
64	(1,1,0,32KB) (8,3,$\mathrm{L_2}$,512KB) (8,38,$\mathrm{L_3}$,8MB) (1,∞,$\mathrm{L_4}$,∞)

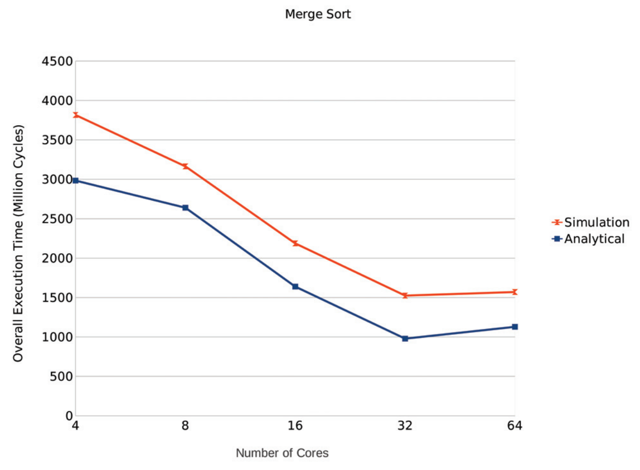

Figure 2. Merge Sort:simulation and analytical results for input size of 36 million integer numbers.

- Convex Hull: In the divide-and-conquer algorithm, merge stages and sequential execution have similar time complexity, so the parallel time is as follows:

$$Exec(n,d) = \sum_{i=0}^{h}(\frac{n}{K^i}\;\log\frac{n}{K^i})S_i$$

Finally, according to these results, it can be implied that our model and the simulation have the same behavior and also the model is able to approximate the behavior of the algorithm.

8. Conclusion and Future works

In this paper we have proposed a model for multi-core computing which is an extension of Multi-BSP model. Our model is called Multi-DaC because it maps divide-and-conquer algorithms on cache hierarchy of multicores. Lower bounds for communication and synchronization costs are provided by the model. In experiments we have studied merge sort and convex hull problems and evaluated them by simulated hardware. The results proved the correctness of our computational model.

Multi-DaC model is an effort to ease the programming on the new emerging multicore architectures. It facilitates implementation of one of the most broad applicable of algorithms i.e. divide-and-conquer. In this regards, we will implement divide-and-conquer skeleton to provide higher-level programming to hide complexity of efficient low-level programming and enhance portability of the code. We are going to develop the model by considering to issues such as load-balancing, false-sharing, thread's overhead and using techniques in implementation like work-stealing, prefetching and lightweight tasking.

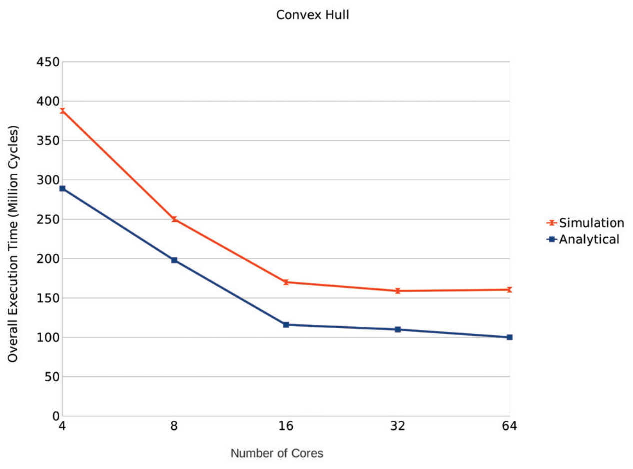

Figure 3. Convex Hull:simulation and analytical results for input size of 1.5 million points.

Acknowledgments

The two first authors of this paper share the same role in preparing the paper.

References

[1] A. Aggarwal and S. Vitter, Jeffrey. The input/output complexity of sorting and related problems. *Commun. ACM*, 31:1116–1127, September 1988. ISSN 0001-0782.

[2] B. Alpern, L. Carter, E. Feig, and T. Selker. The uniform memory hierarchy model of computation. *Algorithmica*, 12:72–109, 1994.

[3] L. Arge, M. T. Goodrich, M. Nelson, and N. Sitchinava. Fundamental parallel algorithms for private-cache chip multiprocessors. In *Proceedings of the twentieth annual symposium on Parallelism in algorithms and architectures*, SPAA '08, pages 197–206, New York, NY, USA, 2008. ACM.

[4] G. E. Blelloch, R. A. Chowdhury, P. B. Gibbons, V. Ramachandran, S. Chen, and M. Kozuch. Provably good multicore cache performance for divide-and-conquer algorithms. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '08, pages 501–510, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.

[5] R. Chowdhury, F. Silvestri, B. Blakeley, and V. Ramachandran. Oblivious algorithms for multicores and network of processors. In *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pages 1–12, april 2010.

[6] D. E. Culler, R. M. Karp, D. Patterson, A. Sahay, E. E. Santos, K. E. Schauser, R. Subramonian, and T. von Eicken. Logp: a practical model of parallel computation. *Commun. ACM*, 39:78–85, November 1996.

[7] W. F. Eddy. A new convex hull algorithms for planar sets. *ACM Trans. Math. Software*, 3:398–403, 1977.

[8] S. Fortune and J. Wyllie. Parallelism in random access machines. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, STOC '78, pages 114–118, New York, NY, USA, 1978. ACM.

[9] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran. Cache-oblivious algorithms. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:285, 1999.

[10] R. L. Graham. An efficient algorithm for determining the convex hull of a finite point set. *Info. Proc. Letters*, 1:132–133, 1972.

[11] R. Neapolitan and K. Naimipour. *Foundations of Algorithms*. Jones and Bartlett Publishers, 2011.

[12] F. P. Preparata and S. J. Hong. Convex hulls of finite sets of points in two and three dimensions. *Commun. ACM*, 20:87–93, February 1977.

[13] V. Ramachandran. Qsm: A general purpose shared-memory model for parallel computation. In S. Ramesh and G. Sivakumar, editors, *Foundations of Software Technology and Theoretical Computer Science*, volume 1346 of *Lecture Notes in Computer Science*, pages 1–5. Springer Berlin / Heidelberg, 1997.

[14] J. Renau, B. Fraguela, J. Tuck, W. Liu, M. Prvulovic, L. Ceze, S. Sarangi, P. Sack, K. Strauss, and P. Montesinos. Sesc simulator, 2005. URL `http://sesc.sourceforge.net`.

[15] J. Savage. Extending the hong-kung model to memory hierarchies. In D.-Z. Du and M. Li, editors, *Computing and Combinatorics*, volume 959 of *Lecture Notes in Computer Science*, pages 270–281. Springer Berlin / Heidelberg, 1995.

[16] J. E. Savage and M. Zubair. A unified model for multicore architectures. In *Proceedings of the 1st international forum on Next-generation multicore/manycore technologies*, IFMT '08, pages 9:1–9:12, New York, NY, USA, 2008. ACM.

[17] L. G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33:103–111, August 1990.

[18] L. G. Valiant. A bridging model for multi-core computing. *Journal of Computer and System Sciences*, 77(1):154 – 166, 2011.