Comparative Analysis of Various Cost Models on the basic of Certain Parameters

Khari A. Armih
Al-zawiya College of Computer Technology
Khari.armih@gmail.com

Moktar M. Lahrashe
Al-zawiya College of Computer Technology
mlahrashe@gmail.com

Abstract

With the growth in heterogeneity, the current focus of designing parallel performance cost models is on providing low-level details of parallel execution to the programs to enable resource-aware partitioning and dynamic load balancing procedures, in particular, for heterogeneous parallel architectures.
This Paper presents a survey of a classification of current and emerging cost models for parallel and distributed environments as well as algorithmic skeletons, and addressing major challenges such as complexity, target architectures, Optimisation and Skeleton-based Approachs.

Keyword: Parallel, heterogeneous, Algorithmic skeletons, Cost model

1. Introduction

Models of parallel computation play an important role in designing and optimising parallel algorithms and applications. These models assist the developer in understanding all-important aspects of the underlying architecture without knowing unnecessary details. Moreover, parallel computational models ware used to predict the performance of a given parallel program on a given parallel machine.

The common way of predicting the performance of parallel program is to derive a symbolic mathematical formula that describes the execution time of that program. This formula has a set of parameters that usually include the size of program, number of processors, and other hardware and algorithm characteristics that affect the execution time of the program. These parameters will be given by a programmer, benchmarking, or profiling tools.

Skeleton-based and similarly structured frameworks have employed these parallel computational models to predict the performance of the parallel algorithms in the early stages of the design process. Consequently, these computational models can assist and guide scheduling algorithmic skeletons on a wide variety of architectures.

Several parallel computational models had been developed for parallel-distributed systems to guide parallel algorithm designers. Good general surveys of early research are given in [5, 6, 7, 10] and a more recent survey is given in [8].

In this paper, we survey several well-known parallel cost models that have been proposed for parallel and distributed environments as well as algorithmic skeletons.

Finally, this survey doesn’t aim to give a comprehensive survey of cost models, which would be much beyond the scope of this report,

but we classify and discuss essential aspects of the existing performance cost models, and give references for further reading.

2. The Family of PRAM Models

The most widely-used cost model in parallel computing is the Parallel Random Access Machine (PRAM) model [9]. The PRAM model was based on the RAM model [10] of sequential computation. The model consists of a global shared memory and a set of sequential processors that operate synchronously. The model assumes that at each synchronous step, each processor can access any memory location in one unit time regardless of the memory location. The PRAM model provides a useful guide for parallel algorithm designers and thereby allows them to ignore all the architecture details of the underlying hardware and concentrate on application-specific issues.

Despite the useful basis provided by the PRAM model for parallel algorithm design, it cannot reflect all the costs of a real parallel machine. This results in non-portable programs due to a number of assumptions made by the model by ignoring the cost of some parallel activities such as synchronisation, memory contention, and communication latency or bandwidth.

Therefore, several realistic variants of PRAM-based models have been introduced to make PRAM more practical. These variants attempt to account for the cost issues of real parallel machines. For example, models such as Block PRAM (BPRAM) [11], Local-Memory PRAM (LPRAM) [12], and Asynchronous PRAMs [13] seek to include the latency cost with the standard PRAM model.

Another PRAM variant is asynchronous PRAMs that add some degree of asynchrony into the basic PRAM model in order to ease the restriction on processors synchronisation. These models differ in the way of the processors are synchronised. They include the

Asynchronous Parallel Random Access Machine model (APRAM) [14] that addresses the synchronisation assumption of the basic PRAM model to allow asynchronous execution, and the Hierarchical PRAM (HPRAM) model [15], which uses the PRAM model as a sub-model, consists of a collection of synchronous PRAMs that operate asynchronously from each other. Another asynchronous model by Gibbons et al. [16] allows the processors to run in an asynchronous manner.

The CRCW PRAM model [17] and the QRQW PRAM model [18, 19] account for memory locations contention, where the read and write to shared memory locations are done concurrently.

3. BSP and Variants

The Bulk Synchronous Parallel (BSP) model [20, 21] is a parallel computation model that provides a simple way of writing parallel programs for a wide range (architecture-independent) of parallel architectures by offering a bridging model that links software and architecture. In addition, it provides a straightforward way for realistic performance prediction for application design on a variety of different parallel architectures including distributed-memory systems, shared-memory multiprocessors, and networks of workstations. Practically, the BSP model aims to provide a bridge model between the software and hardware.

The BSP model consists of a collection of processors that communicate using message passing. The computations in the BSP model are formulated as a series of super steps. Conceptually, each super step is divided into three stages. In the first stage, all processors concurrently compute using only local data. In the second stage, processors exchange messages with each other. In the third stage, all of the processors execute a barrier synchronisation, after they finished sending and receiving messages.

Compared with the PRAM model the BSP model is more realistic, since it accounts for two cost issues of the real parallel machines, namely communication cost and memory latency cost.

Since BSP programs are based on sequential super steps, the model provides a very straightforward approach to cost estimation by firstly, calculating the cost of each super step, and secondly, calculating the cost of the whole BSP program by summing the cost of the super steps. The cost of each super step in a BSP program is given by:

$$T_{superstep} = w + hg + l$$

Where w reflects the cost of the longest running local computation in any of the processors, l is a constant cost (the cost of the barrier synchronisation) that depends on the performance of the underlying hardware, h is the number of messages sent or received per processor and g captures the measurement of the ability of the communication network to deliver these messages. A number of parallel implementations have been proposed using the BSP model [22, 23, 24, 25];

4. The LogP Model Family

LogP [29, 30] is an architecture-independent parallel computation model for designing and analysing parallel algorithms. It is a model for distributed-memory multiprocessors where processors communicate using message passing. LogP provides a good balance between abstraction and simplicity by using a few parameters to characterise the parallel computers and enabling the user to ignore all unnecessary details.

Like the BSP model, LogP is more realistic, since both models try to capture the communication latency and bandwidth through parameters [31], and both models allow the processors to work in a completely asynchronous manner.

Nevertheless, the LogP model gives a more realistic picture than BSP, since LogP has more control over the machine resources by capturing the communication overhead. Furthermore, LogP can be use in parallel systems that are constructed from a collection of complete computers connected by a communication network.

Conceptually, the LogP model consists of a collection of sequential processors interacting through a communication network by exchanging messages, where each processor has direct access to a local memory. The parallel program is executed in an asynchronous way by all processors in the LogP machine.

The LogP model seeks to capture the communication network cost by describing the parallel computer in terms of four elements:

P: the machine's number of processors.

g: communication bandwidth for short message (gap).

L: communication delay (latency).

o: communication overhead (overhead) .

The latency is an upper bound on the time required to send a message from a source processor to its target processor. The overhead is the fixed amount of time that a processor requires to prepare for sending or receiving a message; during this time, the processor cannot perform other operations. The gap is the minimum time interval between sending two messages on the same processor. The gap is the inverse of the available per-processor communication bandwidth for a short message. Several researchers [31, 32, 33] have shown that the LogP model delivers good and accurate predictions for small messages. A number of different extensions of the classic LogP model have been developed to improve prediction accuracy by addressing different communication network issues:

4.1 LogGP

The LogGP model by Alexandrov et al.[34, 35] is an extension of the basic LogP model. Since LogP facilitates only short-message communication transmission between processors and ignores long messages, the LogGP model extended LogP to provide a simple linear model that can model both short- and long-messages.

Just as in the original LogP model, LogGP is developed for distributed memory multiprocessors, where each processor has access to local memory. The processors work in an asynchronous way and communicate with other processors by point-to-point messages.

LogGP uses the parameters (latency, overhead, gap, and number of processors) that were introduced by the LogP model to characterise communication performance. In addition, it introduces a new additional parameter, Gap per byte, G, which captures the communication bandwidth for long message. Thus, the LogGP model uses 1/g for short message and 1/G for long message. In the LogP model, sending a k bytes message between two processors requires sending [k/w] messages, where w is the underlying message size of the machine. This takes:

$$O + \left(\left\lceil \frac{K}{W} \right\rceil - 1\right) * \max(g, O) + l + O \; cysles$$

While sending everything as a single large message in the LogGP model takes:

$$O + (K - 1) * G + L + O \; cycles$$

4.2 LogGPS

The LogGPS model [36] is a parallel computational model that extends LogGP to include the synchronisation cost. As in the original LogP model, LogGP eliminates the synchronisation cost that is needed in other models such as PRAM and BSP. This elimination might make LogGP not accurate enough, while it ignores the need for synchronisation when sending a long message in programs that use high-level communication libraries such as MPI. The LogGPS model has been proposed to address this shortcoming in LogGP.

Sending a long message between two processors is often performed by sending a small message to the receiver to check if it is ready to receive the original message. The process causes the sender processor to be synchronised with the receiver processor and adds a synchronisation cost to the overhead. Thus, the LogGPS model adds one additional parameter, S, which reflects the message-size threshold for synchronising sends.

4.3 HLogGP

Another extension of the LogGP model is the Heterogeneous LogGP [37] model. HLogGP has been specifically proposed for heterogeneous parallel systems to capture the heterogeneity in both communication networks and computational nodes. Since the underlying architecture of the LogGP model is very similar to the cluster architecture, it is considered an appropriate starting point for developing HLogGP.

The HLogGP model extends LogGP by transforming its scalar parameters into matrices. Conceptually, the parameters for overhead and gap are replaced by vector parameters, and latency and Gap is replaced by matrix parameters. Furthermore, to capture the heterogeneity in the computational nodes, the parameter for the

number of processors is replaced by a computational power vector, which describes the physical features for every node in the system. The model has been shown to deliver an accurate prediction on heterogeneous clusters.

4.4 Other LogP extension

Besides the previously mentioned LogP extensions, other extensions have been proposed that aim to address different issues in communication. We briefly discuss some here:

LogP-HMM [38] is a parallel computational model based on the LogP model. The idea of LogP-HMM is to develop an accurate model that accounts for the impact of both network communication and multilevel memory on the performance of parallel algorithms and applications. Therefore, the LogP HMM model extends LogP with the HMM model [39], where the LogP model deals with network communication and the HMM model addresses the memory hierarchy.

LoGPC [40] is a simple model that extends LogP and its extension LogGP to address another aspect of communication networks. It uses the features of both models to account for short message as well as long message bandwidth. Practically, the LoGPC model is intended to capture the impact of message transmissions of size m.

Parametrised LogP [41] or (pLogP for short) is a slight extension of the LogP and LogGP models. This model can accurately predict the completion time of collective operations in message passing models such as MPI.

Five parameters are used in the pLogP model to characterised the network. Like the LogP model, it uses P as the number of processors and L is the end-to-end latency, but the original parameters o and g are replaced by a function of message size, where o_s (m) and o_r (m) are

the sender and receiver overheads of the message size m, and g(m) is the delay between consecutive message transmissions of size m.

5. HiHCoHP

The HiHCoHP model [42, 43] is a realistic communication model for hyperclusters (multi-level clusters of clusters of processors) with heterogeneous processors. It aims to capture the important features of a real hypercluster such as bandwidth and transmission cost.

The HiHCoHP model is based on several parameters that reflect the heterogeneity of hyperclusters:

• P_i ("computing power"): HiHCoHP considers the computing power as N heterogeneous nodes that may differ in computational power (computation and memory speed).

• (" message processing"): the P_a and P_b set up fixed communication cost is $(\sigma_a^{(k)} + \sigma_b^{(k)})$; and the cost of message packing in P_a is $\pi_a^{(k)}$ and message unpacking in P_b is $\pi_b^{(k)}$.

• $\lambda^{(k)}$ ("network latency"): or the end-to-end latency is the amount that is required to send one packet between the source node and destination node at level-k of the network.

• $\beta^{(k)}$ ("link-bandwidth"): the amount of data that can be sent between two nodes at level-k of the network.

• $k^{(k)}$ ("Network capacity"): the maximum number of packets that can be transmitted at once.

So the total end-to-end communication time of sending p-packet message from node P_a to node P_b is given by:

$$(\sigma_a^{(k)} + \sigma_b^{(k)}) + (\pi_a^{(k)} + \pi_b^{(k)})p + \lambda^{(k)} + \Delta(p)$$

Where $\Delta(p) = (p-1)/\beta^{(k)}$ in a pipeline network, and $\Delta(p) = \lambda^{(k)}(p-1)$ in a store-and-forward network.

6. DRUM

Another type of parallel computational model are architectures-aware cost models. One of the well-known models is the Dynamic Resource Utilisation Model [44, 45] or (DRUM). DRUM is developed to support resource-aware load balancing in a heterogeneous environment such as clusters and hierarchical clusters (clusters of clusters, or clusters of multiprocessors).

DRUM accounts for the capabilities of both network and computing resources. In particular, DRUM is intended to encapsulate information about the underlying hardware, and provide monitoring facilities for hardware capabilities evaluation. Benchmarks are used to assess the capabilities of computational, memory and communication resources.

Each node in the tree structure of the DRUM model has been given a single value called "power", which represents the portion size of the total load that can be assigned to that node based on its processing and communication power.

The power of node n in the DRUM model is calculated as the weighted sum of processing power p_n and communication power c_n:

$$power_n = w_n^{comm} c_n + w_n^{cpu} p_n, \quad w_n^{comm} + w_n^{cpu} = 1$$

7. Skeletons

To improve the performance of parallel applications, performance cost models are associated with algorithmic skeletons to accurately predict the costs of parallel applications. More precisely, the aim of these performance models is to assist the parallel skeletons, either implicitly or explicitly, to guide scheduling on a wide variety of architectures.

This section deals with skeleton-associated performance cost models. Several skeleton-based and similarly structured frameworks have employed performance cost models for various kinds of skeletons. Some of the skeleton-based frameworks employ the well-known cost models and their variants such as the models that were previously mentioned, and others use their own performance prediction tools to estimate the performance of a given program.

Here, we briefly outline the skeleton-based frameworks that employ high-level cost models.

7.1 Darlington's group

Performance models are proposed in [46] for processor farms, divide and conquer (DC), and pipeline skeletons. For example, a performance model has been proposed for a divide and conquer skeleton to provide a prediction of the execution time for given program, which is used to guide resource allocation.

In this model, the total execution time required to solve a problem of size N on P processors is given by:

$$T_{solN} = \sum_{i=1}^{log(p)} \left(T_{\mathrm{divN/2i-1}} + T_{\mathrm{combN/2i}} + T_{comms}\right) + T_{\mathrm{solN/P}}$$

Where T_{divN} is the time to divide a problem of size N, T_{combN} is the time to combine the two results, and T_{comms} is the communication time between processors.

7.2 BSP-based Approaches

Several authors associate the BSP model with algorithmic skeletons for performance optimisation.

For example, Skel-BSP [23, 47] is a subset of P3L that uses an extension of the BSP model called the Edinburgh-Decomposable-BSP model to achieve performance portability for skeletal programming. EdD-BSP extends the BSP model by adding partition and join operations to partition and reunify BSP submachines which allows subset synchronisation as in D-BSP.

Compared to the standard BSP model, EdD-BSP replaces the g parameter with two parameters, which are g∞ and $N_1/2$, and then estimates the cost of two kinds of supersteps:

a) The cost of computational supersteps is given by:

$$T = W + hg\infty(N_{1/2}/h + 1) + L$$

b) The cost of partition and join superstep is given by L

Another BSP-based approach is Bulk-Synchronous Parallel ML (BSML) [151].

BSBML is a functional data parallel language for programming BSP algorithms using a set of high-level parallel primitives. It uses the BSP model to predict the performance of a given program on a wide variety of parallel architectures.

7.3 P3L

P3L uses a variant of the LogP model to predict and optimise program performance on parallel systems. An analytic model is presented in [49] for the basic forms of parallelism to be used by the template-based compiler of the P3L language.

This model is more complex than LogP, since it is intended to capture several hardware features, such as the speed of processor, node architecture, and network bandwidth and latency.

Here we briefly describe the analytical model for the high level template that is related to this work.

The Map construct is implemented on an N dimension grid of processors. The computation time T of input granularity k is given by:

$$T(\mathrm{k}) = k\left(T_{dis}() + T_c \prod_{i=1}^{N} d_i + T_{col}\right)$$

Where:

T_c : seq. computation time.

d_i : data granularity for dimension i.

T_{dis}: data distribution time.

T_{col}: time for collecting results.

7.4 HOPP

The HOPP (Higher-order Parallel Programming) model [50, 51] is a methodology based on the BMF (Bird-Meertens Formalism)[52], where the program is expressed as a composition of higher-order functions.

The HOPP model uses a cost model introduced in [52] to predict the costs of programs. This cost model is implemented as an analyser for calculating the costs of possible implementations for a given program on a given distributed-memory machine.

In the HOPP model, the cost of a program is computed in terms of n steps:

$$cost = \sum_{i=1}^{i=n} C_{pi} + \sum_{i=0}^{i=n-1} C_{i,i+1}$$

where C_{pi} is the cost of phase i which depends on the number of processors and sequential implementation of the functions in that step, and $C_{i,i+1}$ is the cost of communication that may be incurred between step i and step $i + 1$.

7.5 SkelML

SkelML [53] gives performance models for a number of skeletons such as pipeline, farm, and fold Processor Chain skeletons. These models are based on the communication overhead and computation time that are involved in application execution. The skeleton performance models and profiling information help the SkelML compiler to determine useful parallelism.

8. Resource Metrics for Parallel Cost Models

The performance of parallel machines is dependent on the underlying architecture features. These features are referred to as resource metrics that characterise the parallel computational model. Thus, a computational model can be identified by a set of these resource metrics. We now consider some resource metrics that are visible in all parallel computational models.

Number of processors the number of processor in the machine.

Communication Latency is the time needed to transfer a message from one processor to another processor; this depends on both the network topology and technology.

Communication Bandwidth is the amount of data that can be sent within a given time; this is a limited resource in practice and depends on the network interface.

Communication Overhead is the period of time that is needed by the processor for sending and receiving message. The amount of overhead depends on network topology features such as communication protocols.

Computational power Computational power is the amount of work finished by one processor in a given time for a specific task; this value depends on the processor's capabilities and the task being processed.

Synchronous/Asynchronous In a synchronous model, all processors are synchronised after executing each instruction. Processors may run semi asynchronously, where the computations occur asynchronously within each phase and all processors are synchronised at each phase.

Table 1 shows how these resource metrics contribute in forming the computational models considered above.

Table 1: Resource metrics for parallel computation

Model	Procs	Latency	Bandwidth	Overhead	Computational Power	Synch Asynch
PRAM	√					Synch
BSP	√	√	√			Semi-synch
LogP	√	√	√	√		Asynch
LogGP	√	√	√	√		Asynch
HLogGP	√	√	√	√	√	Asynch
SkelML	√			√		Asynch
P3L	√	√	√	√	√	Asynch
Ske-BSP	√	√	√			Semi-synch
HOPP	√	√	√	√	√	Asynch

9. Conclusion

Here, we have carried out the comparative systematic study of some software cost estimation models in conjunction with their relevant techniques. Although it would be very difficult to say, which model is preeminent as it is vastly based on the size of software and certain other underlying hardware specifications. We claim performance cost models that based on architectural details of a parallel machine to provide cost estimation of a given program on a given machine, provides a reasonable trade-of between the accuracy and simplicity needed for our heterogeneous skeletons.

Reference

[1] P. B. Gibbons, Y. Matias, and V. Ramachandran. The QRQW PRAM: Accounting for Contention in Parallel Algorithms. In Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '94, pages 638-648, Philadelphia, PA, USA, 1994. Society for Industrial and Applied Mathematics.

[2] P. Gibbons, Y. Matias, and V. Ramachandran. E_cient Low-Contention Parallel Algorithms. In The 1994 ACM Symp. on Parallel Algorithms and Architectures, pages 236-247, 1994.

[3] L. G. Valiant. A Bridging Model for Parallel Computation. Commun. ACM, 33:103-111, August 1990.

[4] D. B. Skillicorn, J. M. D. Hill, and W. F. McColl. Questions and Answers about BSP. Scientific Programming, 6(3):249-274, 1997.

[5] J. M. D. Hill, B. McColl, D. C. Stefanescu, M. W. Goudreau, K. Lang, S. B. Rao, T. Suel, T. Tsantilas, and R. H. Bisseling. BSPlib: The BSP Programming Library. Parallel Computing, 24(14):1947-1980, 1998.

[6] A. Zavanella. Skel-BSP: Performance Portability for Skeletal Programming. In Proceedings of the 8th International Conference on High-Performance Computing and Networking, HPCN Europe 2000, pages 290-299, London, UK, UK, 2000. Springer-Verlag.

[7] M. Goudreau, K. Lang, S. Rao, T. Suel, and T. Tsantilas. Towards Efficiency and Portability: Programming with the BSP Model. In Proceedings 216 Bibliography of the Eighth

Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA '96, pages 1-12, New York, NY, USA, 1996. ACM.

[8] A. Goldchleger, A. Goldman, U. Hayashida, and F. Kon. The implementa-tion of the BSP Parallel Computing Model on the InteGrade Grid Middle-ware. In Proceedings of the 3rd International Workshop on Middleware for Grid Computing, MGC '05, pages 1-6, New York, NY, USA, 2005. ACM.

[9] P. de la Torre and C. Kruskal. Submachine Locality in the Bulk Syn-chronous Setting. In Luc Boug, Pierre Fraigniaud, Anne Mignotte, and Yves Robert, editors, Euro-Par'96 Parallel Processing, volume 1124 of Lecture Notes in Computer Science, pages 352-358. Springer Berlin / Heidelberg, 1996. 10.1007/BFb0024723.

[10] B. H. Juurlink and H. G. Wijsho_. The E-BSP model: Incorporating General Locality and Unbalanced Communication into the BSP Model. In Luc Boug, Pierre Fraigniaud, Anne Mignotte, and Yves Robert, editors, Euro-Par'96 Parallel Processing, volume 1124 of Lecture Notes in Computer Science, pages 339-347. Springer Berlin Heidelberg, 1996.

[11] L. G. Valiant. A Bridging Model for Multi-core Computing. In Proceedings of the 16th annual European symposium on Algorithms, ESA '08, pages 13-28, Berlin, Heidelberg, 2008. Springer-Verlag.

[12] D. Culler, R. Karp, D. Patterson, A. Sahay, K. E. Schauser, E. Santos, R. Subramonian, and T. von Eicken. LogP: Towards A Realistic Model of Parallel Computation. In Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and

Practice of Parallel Programming, PPOPP '93, pages 1-12, New York, NY, USA, 1993. ACM.

[13] 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(11):78-85, November 1996.

[14] G. Bilardi, K. T. Herley, A. Pietracaprina, G. Pucci, and P. Spirakis. BSP vs LogP. In Proceedings of the Eighth Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA '96, pages 25-32, New York, NY, USA, 1996. ACM.

[15] T. Hoeer, L. Cerquetti, and F. Mietke. A Practical Approach to the Rating of Barrier Algorithms Using the LogP Model and Open MPI. In Proceedings of the 2005 International Conference on Parallel Processing Workshops, ICPPW '05, pages 562-569, Washington, DC, USA, 2005. IEEE Computer Society.

[16] D. Culler, L. T. Liu, R. P. Martin, and C. Yoshikawa. LogP Performance Assessment of Fast Network Interfaces, 1996.

[17] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman. LogGP: incorporating long messages into the LogP model | one step closer towards a realistic model for parallel computation. In Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures, SPAA '95, pages 95-105, New York, NY, USA, 1995. ACM.

[18] A. Alexandrov, M. F. Ionescu, K. E. Schauser, and C. Scheiman. LogGP: Incorporating Long Messages into the LogP Model for Parallel Computation. Journal of Parallel and Distributed Computing, 44(1):71-79, 1997.

[19] F. Ino, N. Fujimoto, and K. Hagihara. LogGPS: A Parallel Computational Model for Synchronization Analysis. In Proceedings of the Eighth ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming, PPoPP '01, pages 133-142, New York, NY, USA, 2001. ACM.

[20] J. L. Bosque and L. Pastor. A Parallel Computational Model for Heterogeneous Clusters. IEEE Trans. Parallel Distrib. Syst., 17:1390-1400, December 2006.

[21] Z. Li, P. H. Mills, and J. H. Reif. Models and Resource Metrics for Parallel and Distributed Computation. In Proceedings of the 28th Hawaii International Conference on System Sciences, HICSS '95, pages 51-, Washington, DC, USA, 1995. IEEE Computer Society.

[22] J. S. Vitter and E. A. M. Shriver. Optimal disk I/O with Parallel Block Transfer. In Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing, STOC '90, pages 159-169, New York, NY, USA, 1990. ACM.

[23] C. A. Moritz and M. Frank. LoGPC: Modeling Network Contention in Message-Passing Programs. IEEE Trans. Parallel Distrib. Syst., 12(4):404-415, 2001.

[24] T. Kielmann, H. E. Bal, and S. Gorlatch. Bandwidth-E_cient Collective Communication for Clustered Wide Area Systems. In Parallel and Distributed Processing Symposium, 2000. IPDPS 2000. Proceedings. 14th International, pages 492 -499, 2000.

[25] F. Cappello, P. Fraigniaud, B. Mans, and A. L. Rosenberg. HiHCoHP: Toward a Realistic Communication Model for Hierarchical HyperClusters of Heterogeneous Processors. In

Proceedings of the 15th International Parallel & Distributed Processing Symposium, IPDPS '01, pages 42-, Washington, DC, USA, 2001. IEEE Computer Society.

[26] A. L. Rosenberg. Sharing PartitionableWorkloads in Heterogeneous NOWs: Greedier Is Not Better. In Proceedings of the 3rd IEEE International Conference on Cluster Computing, CLUSTER '01, pages 124-, Washington, DC, USA, 2001. IEEE Computer Society.

[27] K. D. Devine, E. G. Boman, R. T. Heaphy, B. A. Hendrickson, J. D. Teresco, J. Faik, J. E. Flaherty, and L. G. Gervasio. New Challenges in Dynamic Load Balancing. Appl. Numer. Math., 52(2-3):133{152, February 2005.

[28] J. Faik, J. D. Teresco, K. D. Devine, J. E. Flaherty, and L. G. Gervasio. A Model for Resource-aware Load Balancing on Heterogeneous Clusters. Technical Report CS-05-01, Williams College Department of Computer Science, 2005.

[29] J. Darlington, A. J. Field, P. G. Harrison, P. H. J. Kelly, D. W. N. Sharp, and Q. Wu. Parallel Programming Using Skeleton Functions. In PARLE '93: Proceedings of the 5th International PARLE Conference on Parallel Architectures and Languages Europe, pages 146-160, London, UK, 1993. Springer-Verlag.

[30] A. Zavanella. Skeletons and BSP: Performance Portability for Parallel Programming. PhD thesis, UNIPI, December 1999.

[31] F. Gava. BSP Functional Programming: Examples of a Cost Based Methodology. In Proceedings of the 8th international conference on Computational Science, Part I, ICCS '08, pages 375-385, Berlin, Heidelberg, 2008. Springer-Verlag.

[32] D. Pasetto and M. Vanneschi. Machine-independent analytical models for cost evaluation of template-based programs. In PDP, pages 485{492, 1997.

[33] R. Rangaswami. Compile-Time Cost Analysis for Parallel Programming. In Proceedings of the Second International Euro-Par Conference on Parallel Processing-Volume II, Euro-Par '96, pages 417-421, London, UK, 1996. Springer-Verlag.

[34] R. Rangaswami. A Cost Analysis for a Higher-order Parallel Programming Model. PhD Thesis. University of Edinburgh, Department of Computer Science, 1996.

[35] R. S. Bird. Algebraic Identities for Program Calculation. Comput. J., 32(2):122-126, April 1989.

[36] D. B. Skillicorn and W. Cai. A Cost Calculus for Parallel Functional Programming. J. Parallel Distrib. Comput., 28(1):65{83, 1995.

[37] T. A. Bratvold. Skeleton-Based Parallelisation of Functional Programs. PhD thesis, Heriot-Watt University, November 1994.

[38] D. G. Lowe. Object Recognition from Local Scale-Invariant Features. In ICCV '99: Proceedings of the International Conference on Computer Vision-Volume 2, page 1150, Washington, DC, USA, 1999. IEEE Computer Society.

[39] D. G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. Int. J. Comput. Vision, 60(2):91-110, 2004.

[40] S. Gupta. Performance Analysis of GPU Compared to Single-Core and Multi-Core CPU for Natural Language Applications.

IJACSA - International Journal of Advanced Computer Science and Applications, 2(5):50-53, 2011.

[41] M. McCool and S. D. Toit. Metaprogramming GPUs with Sh. AK Peters Ltd, 2004.

[42] AMD Corporation. ATI Stream Computing User Guide, Version 2.01. Technical report, 2010.

[43] K. Komatsu, K. Sato, Y. Arai, K. Koyama, H. Takizawa, and H. Kobayashi. Evaluating Performance and Portability of OpenCL Programs. In The Fifth International Workshop on Automatic Performance Tuning, UC Berkeley - CITRIS, Sutardja Dai Hall, Berkeley, CA 94720, USA, June 2010.

[44] M. M. Baskaran, U. Bondhugula, S. Krishnamoorthy, J. Ramanujam, A. Rountev, and P. Sadayappan. A Compiler Framework for Optimization of A_ne Loop Nests for GPGPUs. In Proceedings of the 22nd Annual International Conference on Supercomputing, ICS '08, pages 225-234, New York, NY, USA, 2008. ACM.

[45] S. Lee, S. Min, and R. Eigenmann. OpenMP to GPGPU: A Compiler Framework for Automatic Translation and Optimization. SIGPLAN Not, 44(4):101-110, February 2009.

[46] J. Hoberock and N. Bell. Thrust: A Parallel Template Library @http://www.meganewtons.com, 2009.

[47] CUDPP: CUDA Data Parallel Primitives Library@http://gpgpu.org/developer/cudpp, 2009.

[48] M. Steuwer, P. Kegel, and S. Gorlatch. SkelCL - A Portable Skeleton Library for High-Level GPU Programming. In

Proceedings of the 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and PhD Forum, IPDPSW '11, pages 1176-1182, Washington, DC, USA, 2011. IEEE Computer Society.

[49] A. D. Malony, S. Biersdor_, S. Shende, H. Jagode, S. Tomov, G. Juckeland, R. Dietrich, D. Poole, and C. Lamb. Parallel Performance Measurement of Heterogeneous Parallel Systems with GPUs. In Proceedings of the 2011 International Conference on Parallel Processing, ICPP '11, pages 176-185,

Washington, DC, USA, 2011. IEEE Computer Society.

[50] Y. Ogata, T. Endo, N. Maruyama, and S. Matsuoka. An Efficient, Model-based CPU-GPU Heterogeneous FFT Library. In Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on, pages 1-10, April 2008.

[51] C. Yang, F. Wang, Y. Du, J. Chen, J. Liu, H. Yi, and K. Lu. Adaptive Optimization for Petascale Heterogeneous CPU/GPU Computing. In Proceedings of the 2010 IEEE International Conference on Cluster Computing, CLUSTER '10, pages 19-28, Washington, DC, USA, 2010. IEEE Computer Society.

[52] V. Strassen. Gaussian Elimination is not Optimal. Numerische Mathematik, 14(3):354-356, 1969.

[53] M. Aldinucci, M. Danelutto, and P. Teti. An advanced Environment Supporting Structured Parallel Programming in Java. Future Gener. Comput. Syst., 19(5):611{626, July 2003.