Incorporating Latency in Heterogeneous Graph Partitioning

Eric Aubanel and Xiaochen Wu
Faculty of Computer Science
University of New Brunswick
Fredericton, N.B.
Canada, E3B 5A3
{aubanel, Xiaochen.Wu}@unb.ca

Abstract

Parallel applications based on irregular meshes make use of mesh partitioners for efficient execution. Some mesh partitioners can map a mesh to a heterogeneous computational platform, where processor and network performance may vary. Such partitioners generally model the computational platform as a weighted graph, where the weight of a vertex gives relative processor performance, and the weight of a link indicates the relative transmission rate of the link between two processors. However, the performance of a network link is typically characterized by two parameters, bandwidth and latency, which cannot be captured in a single weight. We show that taking into account the network heterogeneity of a computational resource can significantly improve the quality of a domain decomposition obtained using graph partitioning. Furthermore, we show that taking into account bandwidth and latency of the network links is significantly better than just considering the former. This work is presented as an extension to the PaGrid partitioner, and includes a model for estimated execution time, which is used as a cost function by the partitioner but could also be used for performance prediction by application-oriented schedulers.

1. Introduction

Mesh-based parallel programs, which employ a graph to describe the computation and communication requirements of an underlying mesh, form a large part of the increasing role of simulation in science and engineering. Heterogeneous computational platforms are increasingly unavoidable, with the increasing hierarchical levels of computational resources, from multicore processors to clusters of multiprocessors and mini-grids (clusters of clusters). Heterogeneous graph partitioners are vital in order to ensure the efficient execution of mesh-based programs on these platforms.

Given a mesh-based application and a computational platform, the combinatorial mesh partitioning problem can be defined as follows. A weighted graph $G = (V, E)$, consisting of a set of vertices, V, and a set of edges, E, can be used to represent the mesh, and a weighted graph $S = (P, C)$, consisting of a set of processors, P, and a set of connections, C, can be used to represent the platform. The mesh partitioning problem is to map V onto P, that is $\pi : V \longrightarrow P$, such that a specified cost function is minimized.

Existing heterogeneous mesh partitioners, *i.e.* those that partition and map meshes onto heterogeneous platforms, include PART [1], JOSTLE [13], MiniMax [6], SCOTCH [8], and PaGrid [3, 15]. The latter partitioner uses a model of the execution time of the application as a cost function, and minimizes the maximum estimated execution time, in contrast to most partitioners (except for MiniMax and PART), which minimize a measure of total communication cost. It was shown in [15] that PaGrid produces partitions with up to 60% lower estimated execution times than METIS [4], a commonly used homogeneous partitioner, and JOSTLE [13]. Further discussion of these partitioners can be found in [3, 15].

The importance of platform heterogeneity has also been considered in recent work on hierarchical resource-aware dynamic load balancing [2]. The execution environment is represented by a tree, and the power of each node is computed as the weighted sum of processing power and communication power. The power of each node can then be incorporated into load balancing algorithms.

In this paper we examine the importance of incorporating latency into a communication model for heterogeneous mesh partitioning, in the context of the PaGrid partitioner. PaGrid is a multilevel graph partitioner that minimizes an estimate of the execution time of the application. The initial version of PaGrid [3] minimized total communication cost during the refinement phase, and added a load balancing stage based on estimated execution time. This provided promising results, but the partitioner itself was unstable. PaGrid was then redesigned to use estimated execution time as a cost function at all levels of refinement [15].

The outline of the paper is as follows. In Section 2, the estimated execution time formula of PaGrid and the improved formula, incorporating latency, are introduced. In Section 3, the impact of this improvement on the performance of PaGrid and on the estimated execution times of the partitions are presented. Finally, Section 4 contains a concluding discussion.

2. Estimated Execution Time Metric

PaGrid uses multilevel graph partitioning, which has been proven compared to other methods [10]. In this multilevel graph partitioning process of PaGrid, the mesh graph is first repeatedly coarsened using a matching heuristic, modified heavy edge matching [4], until the number of vertices equals the number of processors. Afterward, each processor is randomly assigned one vertex, the estimated execution time t_p of each processor is computed (Equation 2), and the assignment is modified in order to decrease the total execution time over all processors. This assignment is redone at the beginning of each uncoarsening stage, where subdomains are swapped between processors to minimize total execution time $t = \sum_{p \in P} t_p$. PaGrid then proceeds to the refinement phase, where it repeatedly uncoarsens the coarser graph in stages until the finest graph is reached. At each uncoarsening stage, the partition is refined by moving vertices between subdomains in order to minimize the maximum estimated execution time of all processors $t_{max} = \max_{p \in P} t_p$ and balance the t_p of all processors. To accomplish this the change (gain) of t_p is computed for each move of boundary vertices between subdomains. These gains are sorted and filtered to make sure that only advantageous moves are made. More details can be found in [14, 15].

We now introduce PaGrid's estimated execution time formula, discuss its limitations, then add a new term to account for communication latency.

2.1. PaGrid's Original Estimated Execution Time Model

The estimated execution time cost function used in PaGrid is given by:

$$t_p = t^p_{comp}|\pi_p| + \sum_{v \in \pi_p} \sum_{r \in P} |E_r(v)| t^{(p,r)}_{comm} \qquad (1)$$

$$t_p \approx |p| \Big(|\pi_p| + \sum_{v \in \pi_p} \sum_{r \in P} |E_r(v)| R_{p,r} \Big) \qquad (2)$$

where $|p|$ is the relative computational power slowdown of processor p compared to the fastest processor; π_p is the set of vertices that are mapped to processor p and $|\pi_p|$ is the total weight of these vertices; $|E_r(v)|$ represents the sum of edge weights from vertex v to vertices assigned to processor r, $E_r(v) = \{(v,u) | u \in \pi_r\}$. $R_{p,q}$ is given by $t^{(p,q)}_{comm}/t^p_{comp}$, where $t^{(p,q)}_{comm}$ is the communication cost per vertex between p and q, and t^p_{comp} is the computation cost per vertex of p. This ratio is pre-computed $\forall (p,q) \in P^2$ from the given R_{ref}, p_{from}, and p_{to} for a reference link obtained from user input:

$$R_{p,q} = R_{ref} \times \frac{|p_{from}|}{|(p_{from}, p_{to})|} \times \frac{|(p,q)|}{|p|} \qquad (3)$$

where $|(p,q)|$ is the sum of weights of the shortest path from p to q, and represents the transfer time per vertex relative to that of the fastest link.

We discuss two limitations of this model. First, counting edges crossing subdomain boundaries overestimates somewhat the communication required [10]. If a vertex $v \in V$ is assigned to processor p, and has multiple edges with vertices assigned to processor r, the associated communication cost is counted multiple times, whereas transferring vertex v's data to r once is enough. This can be remedied by using the number of boundary vertices instead of the number of edges cut, however this adds significant overhead to the refinement phase. We have found that the slight improvement in the estimated execution times of the partitions does not justify the extra work.

A more significant limitation is that a single weight is used for each edge in platform graph S, whereas two parameters are needed for any reasonable model of network performance: bandwidth and latency.

2.2. Communication Model

The communication time for a message to be sent over a network link depends on a number of factors, including the size of the message, communication protocols, and network contention. However, a simple model which accounts for the size of the message is commonly used:

$$t_{comm} = \lambda + \frac{m}{\beta} \qquad (4)$$

where λ represents latency, m represents the size of the message, and β represents the bandwidth of the link. The first term models the startup time and the second models the transmission time of the message. The tradeoff between these two is an important factor in the design and analysis of parallel algorithms (see, *e.g.* [9]). Failing to account for

Network Type	Band-width β (MB/s)	Band-width Weight w_β	Latency $\lambda(\mu s)$	Latency Weight w_λ
MyriNet 10G	1280	1	2	256
1 Gigabit E.	128	10	80	10240

Table 1. Network Types and Weights

latency can lead to inefficient communication patterns, such as frequent short messages.

In a partitioned mesh, a connection between two vertices that are assigned to different processors results in a communication requirement. The startup time between two processors is given by the latency of the network path between them, if they are assigned adjacent subdomains of the partition, and is equal to zero otherwise. The transmission time between two processors is proportional to the number of edges cut by the subdomain boundary ($|E_r(v)|$) and inversely proportional to the bandwidth of the network path.

In the execution time model of Equation 2 there is only one weight for each link in the platform graph S. A single weight alone cannot account for both latency and bandwidth. Therefore for our new model each link is represented by two weights. Table 1 displays the latency and bandwidth weights for two common switched networks [11]. The weights are assigned as follows. Let w_β represent transmission weight of a link with bandwidth β, β_{max} represent the highest link bandwidth in the network, and w_λ represent the latency weight of a link. Transmission weights are relative to the link with the largest bandwidth:

$$w_\beta = \left\lceil \frac{\beta_{max}}{\beta} \right\rceil \tag{5}$$

In this paper we assume that the data that needs to be communicated per vertex is 10 Bytes long. This can easily be modified for each application. The latency weight of a link is given by:

$$w_\lambda = \left\lceil \frac{\lambda}{10/\beta_{max}} \right\rceil \tag{6}$$

The new execution time formula is obtained by adding a new term $l_{p,r}$ representing startup time between processors p and r to Equation 1:

$$t_p = t^p_{comp}|\pi_p| + \sum_{v\in\pi_p}\sum_{r\in P}|E_r(v)|t^{(p,r)}_{comm} + \sum_{r\in P} C_{p,r} l_{p,r} \tag{7}$$

$$C_{p,r} = \begin{cases} 1 & \text{if } \exists(v,w)|w \in \pi_r \wedge v \in \pi_p \wedge r \neq p \\ 0 & \text{otherwise} \end{cases}$$

Factoring t^p_{comp} gives:

$$t_p = t^p_{comp}\Big(|\pi_p| + \sum_{v\in\pi_p}\sum_{r\in P}|E_r(v)|\frac{t^{p,r}_{comm}}{t^p_{comp}} + \sum_{r\in P} C_{p,r}\frac{l_{p,r}}{t^p_{comp}}\Big)$$

$$t_p = t^p_{comp}\Big(|\pi_p| + \sum_{v\in\pi_p}\sum_{r\in P}|E_r(v)|R_{p,r} + \sum_{r\in P} C_{p,r}S_{p,r}\Big) \tag{8}$$

$$S_{p,r} = \frac{l_{p,r}}{t^p_{comp}}$$

Since t^p_{comp} is proportional to $|p|$, t_p can be approximated by:

$$t_p \approx |p|\Big(|\pi_p| + \sum_{v\in\pi_p}\sum_{r\in P}|E_r(v)|R_{p,r} + \sum_{r\in P} C_{p,r}S_{p,r}\Big) \tag{9}$$

In this equation, $R_{p,r}$ and $S_{p,r}$ are both given by Equation 3 above, except that now for the former $|(p,r)|$ is given by the maximum bandwidth weight w_β on the shortest path between p and r, and for the latter $|(p,r)|$ is given by the sum of the latency weights w_λ on the same path. Note how this differs from the previous communication model, where $|(p,r)|$ was given by the sum of the weights on the shortest path from p to r.

Equation 9 was incorporated into PaGrid's algorithm, by replacing the previous cost function (Equation 2). We call the result PaGridL. This change only affects the initial partitioning and refinement phases. In this paper, we also use this equation in order to compare the quality of different partitions.

3. Experimental Results

We have evaluated PaGridL using two criteria: performance of the partitioner itself, and quality of the partitions based on the estimated execution time (Equation 9). The meshes and system graphs used are described in Sections 3.1 and 3.2. The performance of PaGridL is presented in Section 3.3, and the quality of the partitions are analyzed in Section 3.4.

When PaGridL was first tested, we found that its performance was exceptionally bad in some cases. Analysis revealed that the particularly high latency weights (w_λ - see Table 1) caused an increase of the number of moves during the refinement phase. This particularly affected partitions of small meshes and those of low average degree. A move that created a new communication requirement between two processors caused a large increase in estimated execution time, which then resulted in a number of vertex migrations to mitigate this increase. To avoid this problem we experimented with scaling back the latency weights by

Graph	$\lvert V\rvert$	$\lvert E\rvert$	Avg(degree)
auto	448,695	3,314,611	14.77
144	144,649	1,074,393	14.86
ocean	143,437	409,593	5.71
4elt	15,606	45,878	5.88
crack	10,240	30,380	5.93

Table 2. Application Graphs

System	# of processors	Processor weight	Intra-cluster network	Inter-cluster network
32-Homo	32	1	Myrinet 10G	N/A
HS16-2	32	1	Myrinet 10G	gigabit ethernet

Table 3. Chosen Platforms

factors of 100, 50, 10, and 5. In the following we present the results of these experiments, compared with the previous PaGrid algorithm (based on Equation 2), PaGridL with latency weights set to zero, and PaGridL with the full latency weights. In all cases the full latency weight is used in Equation 9 when comparing partitions.

3.1. Graphs

Although the results for five test graphs are presented here, they are representative of a larger number of experiments carried out. Test graphs were taken from the graph archives of METIS [5], JOSTLE [12], and PARTY [7], and all graphs are unweighted (all weights set to 1). Some characteristics of the five chosen graphs are shown in Table 2, where $|V|$ means the number of vertices, $|E|$ means the number of edges, and Avg(degree) means the average degree of all vertices. Results are mainly given for the first four graphs in Table 2, which have varied numbers of vertices, edges, and average degrees.

3.2. Platform Graphs

Results are presented for partitions onto two platform graphs, 32-Homo and HS16-2, as shown in Table 3. 32-Homo represents a fully connected processor cluster with 32 identical processors, and a Myrinet 10G network. HS16-2 represents a grid of two identical clusters of 16 processors and Myrinet 10G networks connected by a single gigabit ethernet link. All partitions are generated using a reference ratio $R_{ref} = 0.25$ (Equation 3).

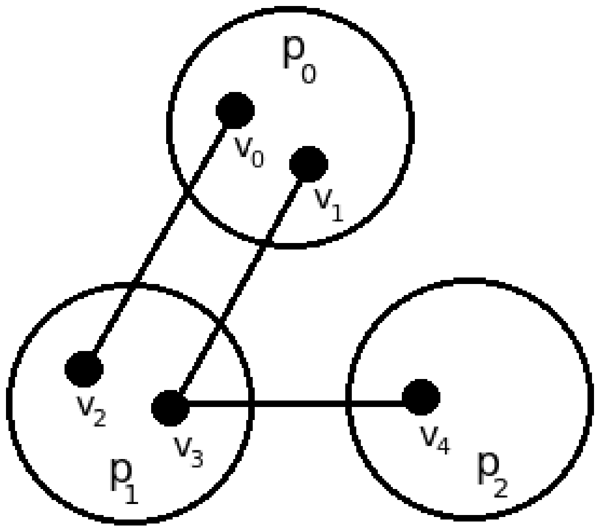

Figure 1. An example of gain update after a move using Equation 9

3.3. Performance

In this section we present a comparison of the run times of PaGrid and PaGridL, which shows the overhead of incorporating latency into the cost function. The main impact of this change is on the computation of the gains of vertex moves in the refinement phase.

In PaGrid, the gains of vertices that are connected to the moved vertex need to be updated after a move. However, PaGridL sometimes has to update the gains of vertices that may have no relationship to the moved vertex.

For example, in Figure 1, when computing the gain of vertex v_0 moving to processor p_1, because there still exists a connection, (v_1, v_3), between p_0 and p_1, the startup time between p_0 and p_1 remains unchanged. However, if vertex v_3 is moved to processor p_2, (v_0, v_2) is the only edge between p_0 and p_1. Therefore, the gain of vertex v_0 moving to processor p_1 changes, and the startup time between p_0 and p_1 needs to be substracted from the gain associated with the migration of vertex v_0 to p_1. As shown in Figure 1, the moved vertex v_3 has no relationship to the affected vertex v_0.

Table 4 shows the performance of the refinement phase and the overall performance for partitioning of two meshes onto HS16-2 grid. In this table, the second column shows PaGrid or PaGridL using different fractions of the original latency weights (latency = 0 and $1/100$, $1/50$, $1/10$, $1/5$ and 100% of full latency weights w_λ). The third and fourth columns show the running time, which is averaged over five runs, in seconds for both the refinement phase and the whole

Graph	Latency Weight	Refinement Phase(s)	Whole Partitioner(s)	# of Moves
144(15)	PaGrid	4.09	6.80	12604
	L = 0	6.37	9.07	11766
	L /= 100	5.12	7.82	11787
	Lty /= 50	5.67	8.38	11597
	L /= 10	5.37	8.07	12211
	L /= 5	5.85	8.55	13677
	L	5.03	7.73	12959
4elt(6)	PaGrid	0.11	0.18	1245
	L = 0	0.14	0.21	884
	L /= 100	0.16	0.24	1204
	Lty /= 50	0.19	0.26	1812
	L /= 10	0.34	0.41	4528
	L /= 5	0.68	0.76	11425
	L	1.19	1.27	22387

Table 4. Performance comparison of PaGrid and PaGridL

partitioner. Finally, the fifth column shows the number of vertex moves for PaGrid and PaGridL when different fractions of the full latency weights are used. One can see that when the full latency weights are used in PaGridL, the performance of PaGridL is poor for graph *4elt*. Similar results were also noted for other meshes, particularly those with low average degree. This is not surprising, as meshes with high vertex degree offer more possibilities for migration of vertices. As noted earlier, this was the reason for the experimentation with scaling back latency weights.

From Table 4, one can see that the performance of PaGridL using full latency weights is worse than that of PaGrid, particularly for the mesh *4elt*. This is not solely because of the overhead in computing gains as noted above, but is also a result of the greater number of moves made during the refinement phase. PaGridL is less than 50% slower than PaGrid if the same number of moves is made. As the latency weights are scaled back the number of moves generally decreases, and the performance of the partitioner increases.

In summary, we have found that when the latency weights used by PaGridL are no higher than $1/5$ of the full latency weights, the execution time for the partitioner is acceptably larger than PaGrid in most cases, especially for high degree graphs. However, when the full latency weights are used, the execution time for the partitioner can be much higher than that of PaGrid.

Graph	Latency Weight	t_{max}
auto(15)	32-Homo	81026.00
	PaGrid	33927.25
	L = 0	33916.00
	L /= 100	34049.00
	L /= 50	35775.00
	L /= 10	33069.00
	L /= 5	32092.50
	L	25485.75
144(15)	32-Homo	59471.75
	PaGrid	39387.00
	L = 0	32066.75
	L /= 100	26590.50
	L /= 50	31449.00
	L /= 10	27904.50
	L /= 5	20174.25
	L	15291.75
ocean(6)	32-Homo	52776.00
	PaGrid	18986.75
	L = 0	21757.00
	L /= 100	16456.25
	L /= 50	21078.00
	L /= 10	20393.25
	L /= 5	15073.50
	L	10472.75
4elt(6)	32-Homo	44837.50
	PaGrid	8679.00
	L = 0	8942.50
	L /= 100	8820.00
	L /= 50	8624.75
	L /= 10	5896.50
	L /= 5	5655.75
	L	5735.75

Table 5. Estimated Execution Times (Equation 9) of Partitions onto HS16-2

Graph	Partitioner	t_{max}	t_{comp}	t_{tran}	t_{late}
auto	PaGrid	17618.25	12773	3821.25	1024
	L=0	17564.50	13363	3241.50	960
	L/=100	17524.75	13400	3164.75	960
	L/=50	17714.00	13167	3395.00	1152
	L/=10	17399.25	13037	3530.25	832
	L/=5	17447.50	12975	3448.50	1024
	L	17210.00	14876	2014.00	320
144	PaGrid	7231.00	3809	1822.00	1600
	L=0	7227.25	3810	1817.25	1600
	L/=100	6984.50	3991	1649.50	1344
	L/=50	6911.50	3952	1679.50	1280
	L/=10	6869.75	3943	1646.75	1280
	L/=5	6814.50	3997	1537.50	1280
	L	6204.00	4906	914.00	384
ocean	PaGrid	5609.50	4365	540.50	704
	L=0	5609.50	4365	540.50	704
	L/=100	5543.25	4404	499.25	640
	L/=50	5545.00	4390	515.00	640
	L/=10	5590.00	4279	607.00	704
	L/=5	5516.25	4326	550.25	640
	L	5278.00	4889	261.00	128
4elt	PaGrid	1230.75	445	81.75	704
	L=0	1230.75	445	81.75	704
	L/=100	1032.75	464	56.75	512
	L/=50	1091.50	450	65.50	576
	L/=10	1073.25	446	51.25	576
	L/=5	1044.00	407	61.00	576
	L	809.25	510	43.25	256

Table 6. Estimated Execution Times (Equation 9) of Partitions onto 32-Homo

3.4. Comparison of Partitions

We first compare in Table 5 partitions onto the HS16-2 graph using PaGrid and PaGridL. As in the previous section we also examine the impact of scaling back latency weights in the cost function used during refinement. We also look at the quality of partitions generated using PaGrid for 32-Homo and mapped to HS16-2, which show the importance of taking into account network heterogeneity when partitioning.

The first column of Table 5 shows the meshes with average degree in parentheses. The second column indicates the partitioner used, including PaGrid, PaGridL with six different latency weights, and 32-Homo means partitions generated using Pagrid for 32-Homo, but mapped to HS16-2. The figures shown are the maximum estimated execution times t_{max} using Equation 9 with full latency weights.

The first observation to be made is that partitions produced without considering the heterogeneity of the network of HS16-2 result in estimated execution times up to five times higher than those produced with PaGrid and HS16-2. Next, one can observe that the difference between partitions from PaGrid and PaGridL with latency weights set to zero are not significant. This is not surprising, since the fact that the former sums transmission weights along a path whereas the latter uses the maximum weight does not make a big difference for the system graph used. Finally, incorporating latency into the cost function reduces the estimated execution time by up to a factor of two. As latency weights are increased the estimated execution times decrease. Furthermore, PaGridL using full latency weights generates the highest partition quality in most cases. However, when the average degree of the graph is small, such as the mesh graph of 4elt, PaGridL using full latency weights does not necessarily generate the best partition quality.

Table 6 shows that incorporating latency not only improves partitions onto heterogeneous platforms, but also on homogeneous platforms. In Table 6, t_{max} decreases as latency increases. The breakdown of t_{max} into computation time t_{comp}, transmission time t_{tran}, and latency t_{late} is also shown in Table 6. One can see that incorporating latency into the cost function decreases transmission time and latency components of t_{max}, not just the latter. This decrease in communication time is a result of a rebalancing of the vertex assignments, which leads to imbalances in vertex weights. These imbalances are not a cause for concern, since what matters is the magnitude of t_{max}, of which computation time is only one component.

From Table 6, one can also see how, for a fixed number of processors, the contribution of latency to the execution time of the application increases as the size of the mesh decreases, reflecting a common observation for the performance of parallel programs employing domain decomposition. One can see the increasing contribution of latency as the number of processors increases in Figures 2 and 3. These figure shows the breakdown of t_{max} for partitions of mesh *144* and *crack* on homogeneous networks from 2 to 64 processors. These graphs illustrate the diminishing return of partitioning these meshes onto more than 32 processors. Ignoring latency in the partitioning of the *crack* mesh would lead one to think erroneously that additional speedup would be possible if more than 32 processors were used.

The partitions in this paper employed a value of $R_{ref} = 0.25$ (Equation 3), which gives a measure of the granularity of the application on a particular platform. For higher values of R_{ref}, corresponding to a finer grain, the importance of latency increases concomitantly with the increasing contribution of communication to the execution time.

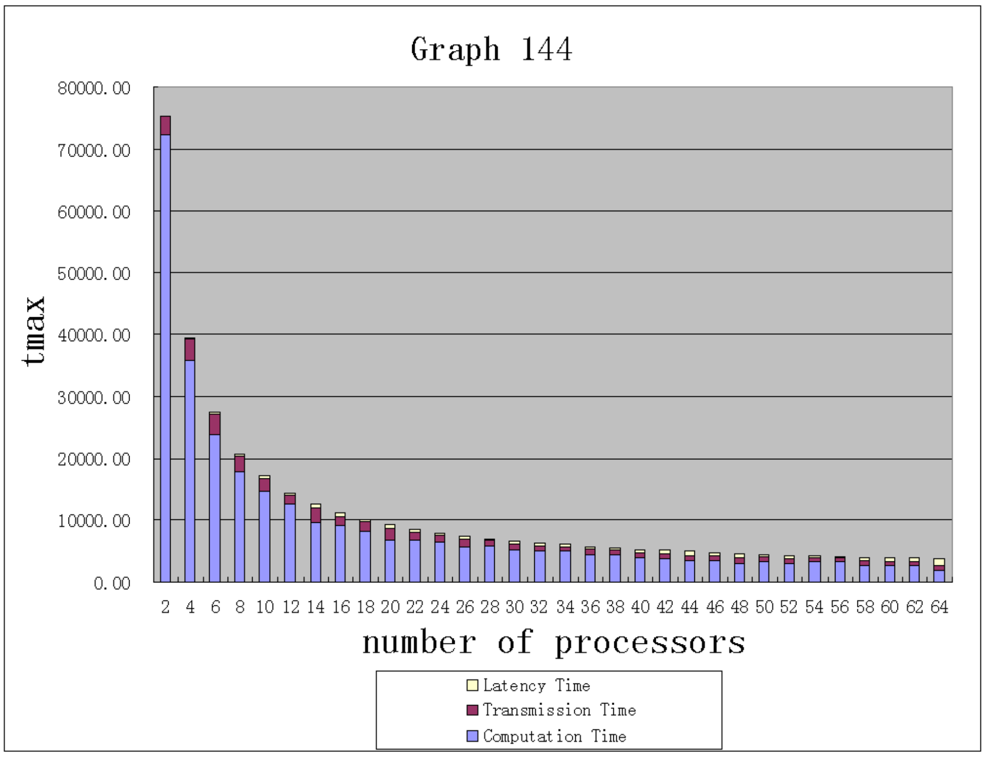

Figure 2. Breakdown of t_{max} for partitions of mesh *144*

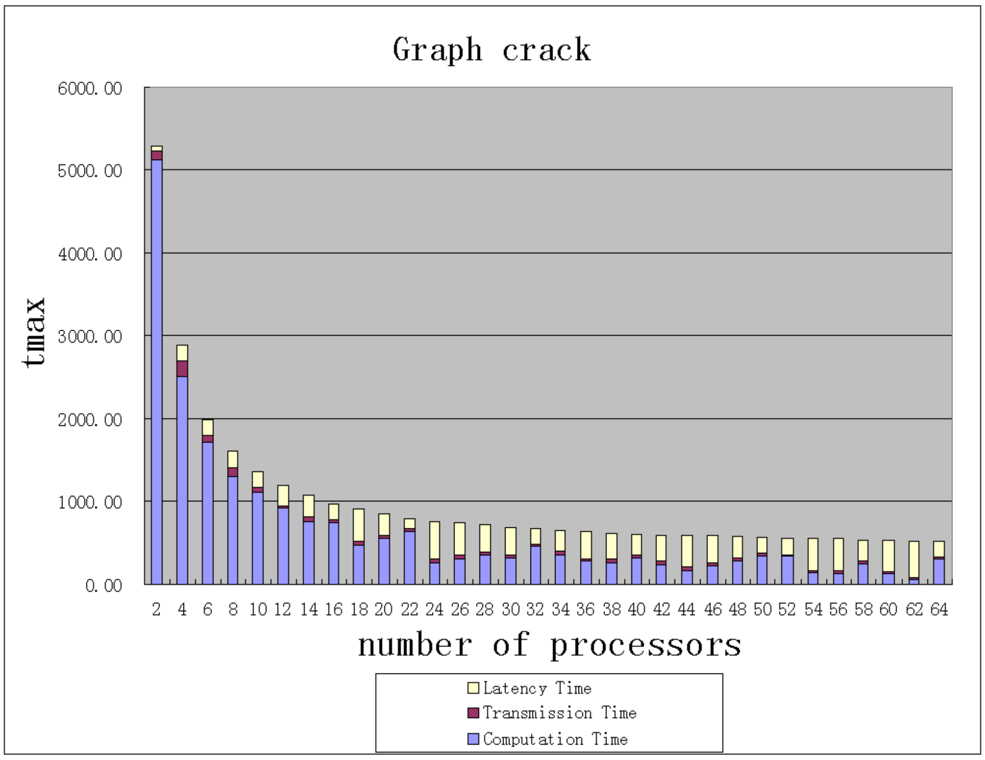

Figure 3. Breakdown of t_{max} for partitions of mesh *crack*

4. Conclusions

There are three main conclusions of this work. First, taking into account the network heterogeneity of a computational resource can significantly improve the quality of a domain decomposition obtained using graph partitioning. Second, taking into account bandwidth and latency of the network links is significantly better than just considering the former. An improvement can even be found for the case of homogeneous resources, a result of the rebalancing of the partition to reduce communication cost. Third, we present an execution time model for mesh-based applications that leads to realistic speedup curves, which could be used for performance prediction by application-oriented schedulers.

This work is presented in the context of the PaGrid graph partitioner, which we extend to incorporate communication latency. We demonstrate the first two conclusions above, on a simple computational grid consisting of two clusters: not considering network heterogeneity can cost a factor of five in execution time of the mesh-based application, and only considering bandwidths can cost a factor of two. We have found that scaling back of the actual weights used to model latency by a factor of five or ten may be necessary to avoid an excessive exploration of the solution space in the refinement phases of the partitioner. Interestingly, even including a small fraction $(1/100)$ of the actual latency weights can lead to noticeable improvement, which continues as the weights are increased. In the case of small meshes and those with low average degree, using large latency weights may lead to poor results, due to the limited solution space exploration possible for these graphs.

We also show that incorporating latency when partitioning onto a homogeneous network can also improve the estimated execution times of the resulting partitions, although the improvements are smaller than for the heterogeneous network considered. We use our execution time model to show how the communication, transmission, and startup time components vary as the number of processors increases.

These results clearly still need to be validated with performance data from actual mesh-based applications. Work is ongoing toward this goal to create benchmark applications and to emulate heterogeneous networks. The cost function is the only difference so far between PaGridL and PaGrid. Performance improvements can still be made, including modifying the filter functions used when selecting moves and consideration of other algorithms for partitioning of the coarsest graph before the refinement phases.

Acknowledgement

This work was funded by a Discovery grant of the Natural Sciences and Engineering Research Council of Canada.

References

[1] J. Chen and V. E. Taylor. Mesh partitioning for efficient use of distributed systems. *IEEE Trans. Parallel and Distributed Systems*, 13(1):67–79, January 2002.

[2] K. Devine and et. al. New challenges in dynamic load balancing. *Appl. Numer. Math.*, 52(2):133–152, 2005.

[3] S. Huang, E. Aubanel, and V. Bhavsar. Pagrid: A mesh partitioner for computational grids. *Journal of Grid Computing*, 4(1):71–88, March 2006.

[4] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359–392, 1998.

[5] G. Karypis and V. Kumar. Metis graph archive. ftp://ftp.cs.umn.edu/dept/users/kumar/graphs. Last accessed July 2006.

[6] S. Kumar, S. Das, and R. Biswas. Graph partitioning for parallel applications in heterogeneous grid environments. *International Parallel and Distributed Processing Symposium*, 2002.

[7] Party graph collection. http://www.uni-paderborn.de/cs/ag-monien/research/part/graphs.html. Last accessed July 2006.

[8] F. Pellegrini and J. Roman. Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. *High-Performance Computing and Networking(HPCN'96 Europe)*, 1067:493–498, 1996.

[9] M. J. Quinn. *Parallel Programming In C With MPI and OpenMP*. McGraw-Hill Companies, Inc, 2004.

[10] K. Schloegel, G. Karypis, and V. Kumar. Graph partitioning for high-performance scientific simulations. In e. a. Dongarra, J., editor, *Sourcebook of Parallel Computing*, pages 491–541. Morgan Kaufmann, 2003.

[11] Top 500 supercomputing sites: http://www.top500.org. Last accessed July 2006.

[12] C. Walshaw. Walshaw graph collection. http://staffweb.cms.gre.ac.uk/ c.walshaw/partition. Last accessed July 2006.

[13] C. Walshaw and M. Cross. Multilevel mesh partitioning for heterogeneous communication networks. *Future Generation Comput. Syst.*, 17(5):601–623, March 2001.

[14] R. Wanschoor. Mesh partitioning for computational grids. *Master's thesis, Faculty of Computer Science, University of New Brunswick, Fredericton, NB, Canada*, 2004.

[15] R. Wanschoor and E. Aubanel. Partitioning and mapping of mesh-based applications onto computational grids. *Fifth IEEE/ACM International Workshop on Grid Computing*, pages 156–162, November 2004.