Performance Modeling for Hierarchical Partitioning in Heterogeneous Multi-Core Environment

S.Y. Chan * T.C. Ling * E. Aubanel †

January 2013

Abstract

Considering application behavior in graph partitioning is an arduous task because of the chicken-and-egg problem: the application behavior depends on how the finite element (FE) mesh is decomposed while achieving load balance requires the knowledge of how the application utilizes the underlying resources. Recent advances in multi-core processors further complicate the endeavor by introducing undeterministic hardware diversity and intra-node contention. As an attempt to quantify performance for partitioning refinement, we propose a model that predicts execution times of iterative mesh-based applications running on heterogeneous multi-core clusters. Apart from considering resource heterogeneity, the model takes into account hierarchical communication characteristics, overlap between computation and communication, as well as performance penalties due to intra-node contention. We present a detailed methodology on how to obtain key parameters from a real system and pinpoint potential pitfalls of conventional approaches in obtaining the parameters. Experiments were conducted using a synthetic application benchmark solving a partial differential equation. Evaluation shows a good agreement between actual time measurement and the performance model.

Keywords: performance model, benchmark, graph partitioning, memory hierarchy, multicore

1 Introduction

Graph partitioning is widely applied in the Finite Element Method (FEM) to support numerical modeling of scientific and engineering computations. FEM describes physical phenomena of interest on bounded domains and is used to solve Partial Differential Equations (PDEs). The domain is discretized into a finite element (FE) mesh and the PDE is transformed to a linear system which is solved using iterative methods. Accurate approximation to the original problem often requires many iterations and a huge number of elements. To this end, FEM is commonly solved on parallel architectures using graph-partitioning algorithms: given $P$ processors with identical capacity, the FE mesh is partitioned into $P$ subdomains such that every processor is assigned an equal number of elements. Each processor executes the same FEM code on its own subdomain. For elements situated at the subdomain boundary, dependencies on external elements necessitate information exchange with neighboring processors. Apart from balancing workload, graph partitioning ensures that these communication costs are kept minimal so as to reduce total execution time.

While conventional graph-partitioning algorithms perform adequately on homogeneous resources, the recent advances in parallel architectures poses new challenges from multiple dimensions. First, the high variation in processor capacity causes computational powers (speed of computation and memory access)

*Faculty of Computer Science and Information Technology, University of Malaya, Kuala Lumpur, Malaysia
†Faculty of Computer Science, University of New Brunswick, Canada
to differ significantly. Inferring the computational powers prior to graph partitioning, however, is not straightforward because the actual processor capacity also depends on how the FEM application exercises the processor’s subsystems. Second, the emergence of multi-core machines further complicates the endeavor by introducing resource contention on the horizontal memory hierarchy. In addition to local area networks, communication now spans across various interconnects ranging from that within chip multiprocessors (CMPs), between CMPs, among compute nodes, and among clusters. The complex dependence on processor capacity and interconnect performance can prevent application developers from harnessing the power of multi-core processors. Essentially, increased parallelism in FEM applications can only be achieved if the following issues are addressed:

1. How to characterize application requirements in the presence of resource heterogeneity and contention?
2. How to use such information to guide graph-partitioning algorithms?

One way to answer these questions is through an analytical model which predicts the performance of the application on target resources prior or during graph partitioning. Although much work has been undertaken to answer Question 1, the proposed metrics [1] have been formulated based on the application side and are independent of the hardware platform. On the other hand, some existing models disregard the impact of heterogeneity and resource contention on multi-core systems [2,3]. In this paper, we attempt to fill the gap by proposing a model for a hypercluster\(^1\) of multi-core processors. In this environment, heterogeneous multi-core processors are grouped into clusters and are interconnected via a hierarchy of networks with varied capacities. Since processors compete with one another for shared resources, we agree with Zhong et al. [2] that time measurement is a better candidate than a combination of parameters to induce the effects of intra-node (within and between CMPs) contention. Our model predicts execution times of the FEM application based on system properties and application characteristics – a white-box approach adopted at Los Alamos National Laboratory [5]. It is only with predicted information available that we may use the performance model to help explore Question 2.

**Motivation.** The motivation of this work originates from R-HierF, our previously proposed hierarchical partitioning framework for heterogeneous multi-core clusters [6]. In R-HierF, the resource capability of an entity (e.g., a core, a compute node, or a cluster) is defined as a weighted sum of varying capabilities in terms of computational speed, network bandwidth, and sustainable memory bandwidth. How an application utilizes these resources is determined through individual weighting factors. We set all the weights to be equal, leaving the weight formulation as future exploration.

As discussed in our previous work, the use of a weighted sum to represent relative capability is controversial. First, the aggregation of multiple terms possesses a self-canceling effect. A fast multi-core processor on a slow network and a slow one on a fast network are impossible to distinguish from two identical constant values. Second, the relations between each term may not be linear. Unfortunately, given inter-dependencies among hardware properties and application characteristics, we conjecture that the correlation is rarely known. Third, the uniform weighting factor can hardly be generalized to applications with differing requirements. Although tuning the weights accordingly is necessary to attain good performance, the major obstacle to estimating application requirement is the undeterministic correlation between parameters. Such correlation is most likely machine and application specific. Therefore we opt for time measurement to consolidate the impact of all parameters and propose a performance model to estimate runtime without the need for actual execution on target machines.

**Contribution.** The contribution of this work is as follows. We develop a model to quantify the performance of a system of heterogeneous multi-core clusters. To achieve higher accuracy, we relax assumptions

\(^1\)Cappello et al. [4] define a hypercluster as an aggregation of clusters into multi-level clusters of... of clusters of processors.
commonly made in the literature: (i) communication routines implemented as synchronous blocking operations, (ii) no overlap of computation and communication in FEM applications, and (iii) no overlap of communications between levels in memory hierarchy, and (iv) network and memory contentions are negligible. In addition, we present detailed methodology of contention-aware benchmarking on a real-system environment. Specifically, we describe how model parameters are measured and highlight potential pitfalls of conventional methodologies in obtaining the parameters. Last, we conduct experimental studies to validate the proposed model on an actual system.

The rest of this paper is organized as follows. Sect. 2 reviews existing performance models for both homogeneous and heterogeneous platforms. Next, we elaborate on our proposed model in Sect. 3. Sect. 4 evaluates the model and describes the methodology used to measure key parameters. We give additional remarks on the experimental results in Sect. 5. Sect. 6 concludes the paper.

2 Related Work

Performance modeling has been studied extensively in the past two decades, leading to abundant computational and communication models proposed to suit technological advances. Previous work in literature can be categorized based on the environment considered: (i) homogeneous vs. heterogeneous platforms, (ii) uniprocessors vs. multiprocessors, (iii) flat networks vs. a hierarchical structure.

Computational models. Axner et al. [7] and Vidal et al. [8] developed analytical models for the parallel sparse Lattice Boltzmann solver running in the environment of homogeneous uniprocessors and flat networks. Defining three sources of efficiency loss, namely, fractional communication overhead, fractional load imbalance overhead, fractional processor speed overhead, Axner et al. [7] claim that performance prediction for other geometries is possible if all the estimated fractional overheads are known.

Bosque and Pastor [9] formulated a scalability model for heterogeneous uniprocessors running on a flat network. The authors focused on isoefficiency which defines how much the workload increment is in proportion to the number of processors for same efficiency. Hoekstra and Sloot [10] proposed a hierarchical resource model and a metric called Grid Speedup to predict performance of a homogeneous computational Grid. The metric is useful to estimate the application grain size for desired speedup in the presence of high communication overheads between parallel systems. While the proposed model is applicable to hierarchical decomposition, the authors presented no empirical result of model validation and of how parameters are measured at real test beds.

Instead of emphasizing individual architectural properties, Lastovetsky and Reddy [11] stressed that time measurements capture the essence of resource heterogeneity better than a combination of parameters. Based upon the execution rate of arithmetic operations, the authors proposed a model for a flat network of uniprocessors with heterogeneous CPU and memory. Unlike traditional models which use single constants to characterize processor speed, they represent the speed as a continuous function of problem size that predicts performance more accurately when tasks do not fit in memory and processor speeds vary with problem size. Notice that the problem size is defined as the amount of data stored and processed during execution. Therefore, the model is applicable only to compute-intensive applications with minimal communication overheads.

Zhong et al. [2] modified the model of Lastovetsky and Reddy [11] to suit a cluster of heterogeneous multi-core nodes. It is assumed that all nodes are of the same architecture but different in terms of memory module and the number of cores per socket. Considering a node rather than a core as the processing unit, the authors argue that the performance of individual cores cannot be modeled independently because processes interfere with one another for shared resources. Hence, using time measurements which embed the costs of arithmetic operations and memory traffic of all cores serve a plausible approach. While the
adapted model provides a more realistic view about multi-core performance, it inherits the limitation of the original model, i.e., the communication overheads are not taken into account. Thus, the model is likely to predict less accurately for FEM applications with boundary exchanges occurring in every iteration.

Communication models. Lastovetsky et al. [12] presented the LMO model to estimate communication times on a flat network of heterogeneous uniprocessors. The motivation of the LMO model is that extension of a homogeneous model which averages values from every pair of processors can lead to inaccurate prediction if the link characteristics differ substantially from one another. The LMO model resolves this issue by considering distinct pairs of processors with different parameters at the expense of higher complexity.

Nasri et al. [13] proposed the PLP model to cater for a hierarchical cluster-based system with heterogeneous uniprocessors. PLP focuses on transmission of fixed-size Ethernet frames. For the sake of simplicity, the authors describe the latency crossing the multi-level networks through a single parameter $L$. The effectiveness of this implicit modeling is questionable as their experimental test bed involved no Ethernet routers but only Layer-2 switches. In fact, the switches together with the processors connected all share the same broadcast domain (a.k.a. virtual LAN), thereby failing to test the validity of the hierarchical model.

The LMO [12,14] and PLP [13] models adopt hardware characteristics to estimate communication performance. Cameron and Ge [15], however, shows that the middleware costs can dominate communication overheads particularly for large message sizes. As an attempt to include software costs in communication modeling, the authors presented the log$_n$P model and its reduction log$_3$P for a cluster of shared-memory processors (SMPs) [15,16]. Derived from Memory logP [17], log$_n$P separates communication into contiguous and non-contiguous costs. The former indicate the costs as the middleware performs implicit communications during message transmission to/from the user buffer, kernel buffer, and local network buffer. The latter signify the additional times for non-contiguous transfer if data are distributed. To reduce complexity, the authors assume that the effect of multiple messages competing for system resources is negligible.

log$_n$P and log$_3$P estimate communication costs incurred on a vertical memory hierarchy for a cluster of SMPs. Communications on a memory hierarchy of multi-core clusters, on the other hand, is far more complicated. Apart from the vertical hierarchy, a multi-core cluster features intra- and inter-node communication on the horizontal memory hierarchy. Chai et al. [18] have shown that approximately 50 percent of messages of typical applications are transferred on the intra-node interconnect. To capture the performance of both vertical and horizontal memory hierarchies on homogeneous multicore clusters, Tu et al. [3] derived the mlog$_n$P model from log$_n$P [15,16] and added parameter $m$ to specify the number of communication levels. Completion times will be determined according to which level the transmission occurs. Similar to log$_n$P, mlog$_n$P ignores the impact of system contention on communication performance. Depending on the memory requirement of the application, the assumption that the intra-node communication is generally faster than inter-node may not hold when increased number of cores are populated for concurrent execution.

Table 1 summarizes the models aforementioned. Closest to our proposal is HiHCoHP, a model for a hypercluster of heterogeneous processors which are grouped in varied-size clusters and connected to networks with differing capabilities [4]. Nevertheless, HiHCoHP considers neither the impact of resource contention nor overlap in communications. Furthermore, among the models reviewed here, only Barker et al. [5] and Kerbyson and Barker [19] take into account non-blocking communication and application-centric modeling, i.e., the application is to be examined and understood instead of being represented as work units (as in [4,10]). Although the latter approach of load characterization reduces complexity, we believe that understanding the application behavior is crucial to accurate performance modeling.
Table 1: Performance models

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Environment</td>
<td>[7,8]</td>
<td>[11]</td>
<td>[10]</td>
</tr>
<tr>
<td>Processor</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Uniprocessor</td>
<td>No</td>
<td>No</td>
<td>N/A</td>
</tr>
<tr>
<td>Multicore</td>
<td>No</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>Environment</td>
<td>Cluster</td>
<td>Cluster</td>
<td>Cluster</td>
</tr>
<tr>
<td>Hierarchical</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Application centric</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Contention modeling</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Communication overlap</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Communication-computation overlap</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Experimental Validation</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

particularly in the presence of contention on multi-core systems.

In this paper, we seek a representation that is independent of operating system and middleware. Because both computational and communication performance can be influenced by the impact of system contention, we treat this aspect as the fundamental requirement for our model. Rather than using additional parameters to characterize the contention, we employ contention-aware benchmarking to reflect the behavior within basic parameters. Additionally, we assume that the communication is implemented using non-blocking asynchronous operations. This is in line with the common practice of calling MPI-Isend and MPI_Irecv primitives in parallel scientific applications. While our model may omit detailed but important features, we hope that our work will stir new directions for future research in this area.

3 The Modeling Framework

This section describes our performance model which estimates computation and communication times in the environment of heterogeneous multi-core clusters. We explain the assumptions and notations followed by the details of the model. The methodology to measure key parameters will be explained in Sect. 4.

3.1 Problem Formulation

Our model is developed based on following assumptions. First, we assume that an FEM application running with graph $G(N)$ of size $N$ is decomposed for a three-level hypercluster $H$ where each cluster comprises varied numbers of multi-core nodes. This environment leads to three levels of communication costs, i.e., level-one overheads incurred on inter-cluster networks, level-two on inter-node networks, and level-three on intra-node interconnects. Further detail about the topology modeling can be found in our previous work [20]. Second, we assume that communications are in the form of non-blocking point-to-point operations. As a consequence, there exist three kinds of overlapping, i.e., (i) overlap between computation and communication, (ii) overlap of communication costs between multiple levels of the memory hierarchy, and (iii) overlap of communication between a processor to all its neighbors. Finally, $H$ can consist of heterogeneous processor and network capabilities. Thus, the partitioner of choice is presumed to be capable of taking into account resource heterogeneity for optimum performance.

Table 2 lists the notations used in Sect. 3.2. The symbols signify parameters related to graph $G$, topology $H$, and system properties. In this work, we use the volume model instead of the standard
Table 2: Notations and definitions

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G(N)$</td>
<td>Graph $G$ of size $N$ where $N$ is the total number of vertices of $G$</td>
</tr>
<tr>
<td>$W$</td>
<td>Total workload associated to graph $G(N)$</td>
</tr>
<tr>
<td>$\triangle$</td>
<td>The number of iterations performed by the application</td>
</tr>
<tr>
<td>$m$</td>
<td>The number of floating-point operations per graph vertex</td>
</tr>
<tr>
<td>$\beta$</td>
<td>The number of bytes sent per communication volume</td>
</tr>
<tr>
<td>$s_i$</td>
<td>The size of subdomain $s_i$ in terms of number of vertices</td>
</tr>
<tr>
<td>$</td>
<td>s_i</td>
</tr>
<tr>
<td>$H$</td>
<td>A hypercluster of multi-core systems</td>
</tr>
<tr>
<td>$Q_1$</td>
<td>The level-1 interconnect, e.g., communications between a pair of clusters on topology $H$</td>
</tr>
<tr>
<td>$Q_2$</td>
<td>The level-2 interconnect, e.g., communications between a pair of multi-core nodes within a cluster of topology $H$</td>
</tr>
<tr>
<td>$Q_3$</td>
<td>The level-3 interconnect, e.g., communications between a pair of processors within a multi-core node of topology $H$</td>
</tr>
<tr>
<td>$f_{Q1}^i$</td>
<td>Level-1 load fraction of processor $i$</td>
</tr>
<tr>
<td>$f_{Q2}^i$</td>
<td>Level-2 load fraction of processor $i$</td>
</tr>
<tr>
<td>$f_{Q3}^i$</td>
<td>Level-3 load fraction of processor $i$</td>
</tr>
<tr>
<td>$d_{Q1}^i$</td>
<td>The number of level-1 neighbors of processor $i$</td>
</tr>
<tr>
<td>$d_{Q2}^i$</td>
<td>The number of level-2 neighbors of processor $i$</td>
</tr>
<tr>
<td>$d_{Q3}^i$</td>
<td>The number of level-3 neighbors of processor $i$</td>
</tr>
<tr>
<td>$b_{Q1}^{ij}$</td>
<td>The amount of communication volume sent from processor $i$ to $j$ over the level-1 interconnect</td>
</tr>
<tr>
<td>$b_{Q2}^{ij}$</td>
<td>The amount of communication volume sent from processor $i$ to $j$ over the level-2 interconnect</td>
</tr>
<tr>
<td>$b_{Q3}^{ij}$</td>
<td>The amount of communication volume sent from processor $i$ to $j$ over the level-3 interconnect</td>
</tr>
<tr>
<td>$t_{Q1}^i$</td>
<td>Time for processor $i$ to execute one floating-point operation on one graph vertex</td>
</tr>
<tr>
<td>$t_{Q2}^i$</td>
<td>Time for processor $i$ to exchange messages with its neighbors via the level-1 interconnect</td>
</tr>
<tr>
<td>$t_{Q3}^i$</td>
<td>Time for processor $i$ to exchange messages with its neighbors via the level-2 interconnect</td>
</tr>
<tr>
<td>$t_{Q1}^{ij}$</td>
<td>Time for processor $i$ to transmit $\beta \cdot b_{Q1}^{ij}$ bytes to processor $j$ via the level-1 interconnect</td>
</tr>
<tr>
<td>$t_{Q2}^{ij}$</td>
<td>Time for processor $i$ to transmit $\beta \cdot b_{Q2}^{ij}$ bytes to processor $j$ via the level-2 interconnect</td>
</tr>
<tr>
<td>$t_{Q3}^{ij}$</td>
<td>Time for processor $i$ to transmit $\beta \cdot b_{Q3}^{ij}$ bytes to processor $j$ via the level-3 interconnect</td>
</tr>
</tbody>
</table>

The edgecut model to approximate the amount of data sent between processors. The volume\(^2\) model is deemed to estimate communication costs more accurate than its edgecut counterpart [22,23].

3.2 The Performance Model

In a homogeneous processor and flat network environment, $T_p(N)$ denotes the execution time to run graph $G(N)$ on $p$ processors, i.e.,

$$T_p(N) = \frac{T_1(N)}{p} + T_{comm}(N) - T_{saving}(N),$$  \hspace{1cm} (1)

where $T_1(N)$ is the sequential time, $T_{comm}(N)$ is the communication time, and $T_{saving}(N)$ is the saving in communication time. If there is no overlap between communication and computation, $T_{saving}(N) = 0$.

Consider a three-level topology $H$ with $C$ homogeneous clusters, $M$ multi-core nodes per cluster, and $p$ processors per node. Adapting the notation used by Hoekstra and Sloot [10], we express the execution time to run graph $G(N)$ on topology $H$ as

$$T_H(N) = \frac{T_1(N)}{pMC} + \max \left\{ Q_1(W, C), Q_2\left(\frac{W}{C}, M\right), Q_3\left(\frac{W}{MC}, p\right) \right\} - T_{saving}(N),$$  \hspace{1cm} (2)

\(^2\)gpmetis 5.0 [21] readily reports total communication volume of a partition on standard output.
where \(Q_1(W, C)\) is the level-one communication of workload \(W\) on \(C\) clusters. Within every cluster, workload \(\frac{W}{C}\) is decomposed among \(M\) multi-core nodes; \(Q_2(\frac{W}{C}, M)\) denotes the level-two communication overhead. Again within each node, workload \(\frac{W}{MC}\) is further decomposed among \(p\) processors; \(Q_3(\frac{W}{MC}, p)\) defines the level-three communication cost. Because communication occurs concurrently on all levels, the highest cost determines the total communication overhead.

Eq. 2 implies identical computational power and therefore equal share of workload. Here, we attempt to address a more complex environment in which (i) the number of nodes per cluster and cores per nodes are varied, (ii) the computing and network capabilities are heterogeneous, and (iii) the FEM application can be decomposed hierarchically into variable-sized subdomains. Given topology \(\mathcal{H}\) with heterogeneous processors and interconnects, the execution time on processor \(i \in \mathcal{H}\) is defined as

\[
t_i = t_i^{\text{comp}} + t_i^{\text{comm}} - t_i^{\text{saving}}.
\]

\(t_i^{\text{comp}}\) is the time required to perform \(m \cdot |s_i|\) floating-point operations on subdomain \(s_i\) for \(\triangle\) iterations, i.e.,

\[
t_i^{\text{comp}} = m \times |s_i| \times t_{i}^{\text{oper}} \times \triangle,
\]

where \(m\) represents the number of flops per vertex and \(t_{i}^{\text{oper}}\) is the average time to execute one flop. We assume that the computation time is linearly proportional to \(N\). Alternatively, \(|s_i|\) can be substituted by \(N \times f_{Q1}^{i} \times f_{Q2}^{i} \times f_{Q3}^{i}\) to embed the notion of hierarchical decomposition. The values of \(f_{Q1}^{i}, f_{Q2}^{i}\), and \(f_{Q3}^{i}\) will depend on how the partitioner makes the decision.

\(t_i^{\text{comm}}\) is the communication time and \(t_i^{\text{saving}}\) is the saving time of processor \(i\). To simplify the development of the model, we let \(t_i^{\text{saving}} = 0\), estimate the performance of blocking operations, and consider overall saving in Eq. 9. Hence, \(t_i^{\text{comm}}\) can be written as a summation of the times spent at all communication levels, i.e.,

\[
t_i^{\text{comm}} = t_i^{Q1} + t_i^{Q2} + t_i^{Q3}
\]

where each term signifies level-one (inter-cluster), level-two (inter-node), and level-three (intra-node) overheads associated to processor \(i\). Specifically, \(t_i^{Q1}\) is the total time spent to exchange a message sized \(\beta \cdot b_{ij}^{Q1}\) between processor \(i\) and every of its neighboring processor \(j\) over the level-one interconnect, i.e.,

\[
t_i^{Q1} = 2 \times \sum_{j=1}^{d_i^{Q1}} (t_{ij}^{\text{datQ1}}) \times \triangle.
\]

\(t_{ij}^{\text{datQ1}}\) constitutes the average time required by processor \(i\) to send \(\beta \cdot b_{ij}^{Q1}\) bytes to processor \(j\) through the level-one interconnect. \(t_{ij}^{\text{datQ1}}\) is a function of message size \(\beta \cdot b_{ij}^{Q1}\) to be determined in Sect. 4.2.3. The term 2 signifies the two-way exchange communication in FEM applications, i.e., processor \(i\) sends and receives data to and from its neighbors to update boundary values. Likewise,

\[
t_i^{Q2} = 2 \times \sum_{j=1}^{d_i^{Q2}} (t_{ij}^{\text{datQ2}}) \times \triangle,
\]

\[
t_i^{Q3} = 2 \times \sum_{j=1}^{d_i^{Q3}} (t_{ij}^{\text{datQ3}}) \times \triangle
\]

equals the time spent in level-two and level-three communication respectively. Again, the number of neighbors as well as the amount of data to send will depend on the partitioning decision and the process-to-processor mapping option.
Since the parallel runtime is determined by the slowest processor, we obtain total execution time of running graph $G(N)$ on topology $\mathcal{H}$ as

$$T_{\mathcal{H}}(N) \approx \max_i \{t_i\} - T_{\text{saving}}(N). \quad (9)$$

$T_{\text{saving}}(N)$ denotes the overall saving times when using non-blocking communications. Combining Eq. 3 and 5 with 9 gives the execution time as

$$T_{\mathcal{H}}(N) \approx \max_i \{t_i\} - T_{\text{saving}}(N)$$

$$\approx \max_i \{t_{i}^{\text{comp}} + t_{i}^{\text{comm}}\} - T_{\text{saving}}(N)$$

$$\approx \max_i \{t_{i}^{\text{comp}} + (t_{i}^{Q1} + t_{i}^{Q2} + t_{i}^{Q3})\} - T_{\text{saving}}(N). \quad (10)$$

### 4 Experimental Evaluation

This section presents empirical evaluation of the proposed model. We elaborate our experimental setup and discuss how key parameters are estimated. Subsequently, the model is validated by comparing the estimation with actual runtimes. To ensure reproducible results, we repeated any measurement for 10 times and computed arithmetic average to obtain the respective parameter value. Since our FEM application behaves identically within every iteration, the time reported here is of a single iteration.

#### 4.1 Configuration

The experimental setup consists of creating unstructured finite element (FE) meshes, configuring the test bed, and estimating memory requirement of our synthetic application benchmark. The individual tasks are described as follows.

**Test instances.** Table 3 lists the properties of the four 2D FE meshes modeling the problem of fluid dynamics. In order for the coordinates of mesh points available to FE computation, we generated the meshes\(^3\) from existing geometry or by projecting a 3D model to a 2D plane. Each mesh is then converted to the corresponding nodal graph $G$ for execution. Compared with dual graphs, nodal graphs correctly reflect the communication requirement of the underlying computations [21].

**Test bed.** Fig. 1 shows the physical topology of our test bed, a heterogeneous cluster with two Wolfdale nodes and two Clovertown nodes. Both the architectures are UMA-based featuring system buses as the interconnects and L2 caches as the last-level cache (LLC). Table 4 summarizes the architectural properties of interest. We installed Open MPI 1.5.3 to facilitate parallel execution. In the case when applications

---

\(^3\)The meshes are available for download at DIMACS10 [24].

---

<table>
<thead>
<tr>
<th>Mesh</th>
<th># Vertices</th>
<th># Edges</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NACA0015</td>
<td>1,039,183</td>
<td>3,114,818</td>
<td>NACA0015 airfoil</td>
</tr>
<tr>
<td>M6</td>
<td>3,501,776</td>
<td>10,501,936</td>
<td>ORENA M6 wing</td>
</tr>
<tr>
<td>AS365</td>
<td>3,799,275</td>
<td>11,368,076</td>
<td>Eurocopter AS365 Dauphin</td>
</tr>
<tr>
<td>NLR</td>
<td>4,163,763</td>
<td>12,487,976</td>
<td>NLR airfoil with flap</td>
</tr>
</tbody>
</table>

---

}\[86\]
optimized for different processor architectures are required to run, we employed the \texttt{--bind-to-core} option and the multiple-programs-multiple-data (MPMD) mode provided by Open MPI 1.5.3. Atop the physical test bed we built an emulated multi-site platform using EHGrid [25]. Fig. 2 shows the configuration file and the hypercluster topology which is composed of two virtual clusters, i.e., the Wolfdale nodes forming one cluster (namely Wolfdale Cluster hereafter) and the Clovertown nodes forming another (Clovertown Cluster).

**Synthetic Application.** We used the \texttt{Poisson} application benchmark\textsuperscript{4} that we developed in our previous work [6] for evaluation purposes. \texttt{Poisson} uses FEM to solve a PDE $-\Delta u = f$ in a computational domain $\Omega \subset \mathbb{R}^2$ with homogeneous boundary conditions. FEM reduces the PDE to a system of linear equations which is solved using the conjugate gradient (CG) method. Let $n$ be the number of free\textsuperscript{5} nodes of an FE mesh on domain $\Omega$. Representing the mesh as a sparse matrix $A$ of size $n \times n$, \texttt{Poisson} adopts the symmetric Compressed Row Storage (CRS) scheme in which the data structures consist of three

\textsuperscript{4}The code is open-source and available upon request (aubanel@unb.ca).

\textsuperscript{5}Free nodes are those lying in domain $\Omega$ rather than on its boundary.
<table>
<thead>
<tr>
<th>Processor Type</th>
<th>Clovertown</th>
<th>Wolfdale</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cores per Node</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>Total Sockets</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>CPU Speed, GHz</td>
<td>2.33</td>
<td>3</td>
</tr>
<tr>
<td>Theoretical Peak Gflop/s per core</td>
<td>9.32</td>
<td>12</td>
</tr>
<tr>
<td>Memory Type</td>
<td>667 MHz DIMM DDR2</td>
<td>800 MHz DIMM DDR2</td>
</tr>
<tr>
<td>Memory bandwidth</td>
<td>21.3 GB/s read, 10.7 GB/s write</td>
<td>25.6 GB/s read, 12.8 GB/s write</td>
</tr>
<tr>
<td>Interconnect</td>
<td>Gigabit Ethernet</td>
<td>Fast Ethernet</td>
</tr>
<tr>
<td>L1 cache size, KB</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>L2 cache size (MB) per die / cores sharing</td>
<td>4 / 2</td>
<td>6 / 2</td>
</tr>
</tbody>
</table>

Arrays, i.e., one vector $AA$ to store the nonzero elements of $A$, one index vector $JA$ to store the column positions of the elements in $A$, and one index vector $IA$ to keep track of the indices corresponding to the beginning of each row in $AA$ and $JA$ [26]. The length of $AA$ and $JA$ is $e$, the number of nonzero elements, whereas the length of $IA$ is $n + 1$. Additionally, the Poisson implementation initializes four vectors each of length $n$ to execute the iterative CG algorithm. Here the index vector $JA$ determines how one of the vectors is accessed during matrix-vector multiplication within the CG operation. Using the graph notation where $e$ corresponds to $|E|$, the number of edges of graph $G$, and $n$ to $|N|$, the number of free nodes of $G$, the minimum memory requirement for Poisson is denoted by

$$
\text{Memory Requirement} \sim ([8 \text{ bytes}] \times |E|) + ([4 \text{ bytes}] \times |E|) + \\
([4 \text{ bytes}] \times (|N| + 1)) + (4 \times [8 \text{ bytes}] \times |N|)
$$

$$
\sim ([12 \text{ bytes}] \times |E|) + ([36 \text{ bytes}] \times |N|), \quad (11)
$$

where the notations of [8 bytes] and [4 bytes] indicate double- and single-precision computations, respectively. Given that the degree of connectivity $|E|/|N|$ for all test meshes used in this work is roughly 2.95, Eq. 11 can be further simplified as

$$
\text{Memory Requirement} \sim ([12 \text{ bytes}] \times 2.95 \times |N|) + ([36 \text{ bytes}] \times |N|)
$$

$$
\sim [71.4 \text{ bytes}] \times |N|
$$

$$
\sim [71 \text{ bytes}] \times |N|. \quad (12)
$$

### 4.2 Measuring Basic Parameters

This section details our methodology to estimate parameter values for application characteristics, single-processor performance, and point-to-point communication of our synthetic application. The validity of these parameters are also discussed to support justification whenever applicable. Note that all values and functions shown here are specific to hardware architectures, application behavior, partitioning techniques, and mapping strategies. The methodology, however, will serve a guide for developers to estimate application performance on multi-core systems in a systematic way.

#### 4.2.1 Application Characteristics

We used OProfile [27] and FPMPI-2 [28] profilers to derive $m$ and $\beta$ respectively. For $m$, the counter event to measure total floating-point operations on our processor architectures is $\text{SIMD\_COMP\_INST\_}$.
Isend 19 12 and fitted the sampling data to approximate shows the results for 3 reduces the prediction error to an acceptable range. shows the execution. We therefore refined our methodology. Instead of assessing prediction we can infer that our approximation fails to account for resource contention during concurrent estimation with maximum error of -62.3 percent in estimating parallel computation time. From such mis-computed the average of $t_i$ times, we derived

$$m = \frac{\text{Total flops}}{\text{Total cores} \times \text{Total graph vertices} \times \Delta}.$$  

(13)

For $\beta$, FPMPI-2 reports the average bytes sent per process through the MPI_Isend routine. With one process mapped onto one core, we calculate $\beta$ as

$$\beta = \frac{\text{Average bytes sent per process} \times \text{Total processes}}{2 \times \text{Total communication volume} \times \Delta}.$$  

(14)

The term 2 signifies the two-way boundary exchange. We repeated the profiling on different machines to ensure that the derived values are constant. For Poisson, $m = 20$ and $\beta = 4$.

4.2.2 Single-Processor Performance

In our proposed model, $t_i^{\text{comp}}$ is governed by $t_i^{\text{oper}}$ which in turn depends not only on CPU speed, LLC size, memory hierarchy, problem size, but also the application’s data layout. To estimate $t_i^{\text{oper}}$ for Poisson, we first conducted sampling by executing the serial version of the application on a single core with synthetic FE meshes created via Triangle [29]. These meshes are generated from a unit square $\Omega = (0, 1) \times (0, 1)$ and are imposed with a range of fixed area constraint values. By measuring the actual computation times, we derived $t_i^{\text{oper}}$ from Eq. 4 and fitted the sampling data to approximate $t_i^{\text{oper}}$ for all mesh sizes.

While the serial execution on one core appears to be justifiable [7, 19], obtaining $t_i^{\text{oper}}$ for multi-core processors is not straightforward. Our initial attempt involving the serial Poisson causes under-estimation with maximum error of -62.3 percent in estimating parallel computation time. From such mis-prediction we can infer that our approximation fails to account for resource contention during concurrent execution. We therefore refined our methodology. Instead of assessing $t_i^{\text{oper}}$ on a single processor, we computed the average of $t_i^{\text{oper}}$ per processor by running the parallel version of Poisson on the Wolfdale and Clovertown node distinctly. Nevertheless, the resultant computation time remains under-estimated with maximum error of -33.1 percent. Upon closer inspection, we discovered that using the real-world FE meshes described in Table 3 reduces the prediction error to an acceptable range.

Fig. 3 shows $t_i^{\text{oper}}$ of Wolfdale and Clovertown as a function of subdomain size with each trendline derived from different approaches, i.e., (i) parallel Poisson with real-world FE meshes depicted as a dashed line, (ii) parallel Poisson with synthetic FE meshes as a broken-and-dashed line, and (iii) serial Poisson with synthetic FE meshes as a broken-with-two-dots-and-dashed line. As seen in the figure, the sets of measurements can be divided into two regions: $t_i^{\text{oper}}$ is constant, and $t_i^{\text{oper}}$ increases with subdomain sizes. The transition is attributed to the effect of memory hierarchy where the LLC becomes filled. Given the LLC size of Wolfdale as 6 MB and Clovertown as 4 MB (see Table 4) and the memory consumption per vertex of 71 bytes (see Eq. 12), we know that data will no longer fit in cache when the subdomain size per core $|s_i| > 10^{4.647}$ bytes for Wolfdale and $|s_i| > 10^{4.470}$ for Clovertown. Here we presume the cache to be equally shared among two cores.

Notice that the cutting point on the x-axis for parallel Poisson is slightly shifted to the left. The early transition is likely to be caused by higher memory requirement to establish additional data structure for parallel execution. The immediate consequence is smaller effective cache size, higher cache miss ratio, and longer waiting time for data to be fetched from memory before computation can begin. This phenomenon explains the mis-prediction in our attempts to measure $t_i^{\text{oper}}$ when using serial Poisson.

As previously discussed, using parallel Poisson with synthetic meshes still causes large discrepancies in estimating $t_i^{\text{oper}}$. Our examination posits that the error is likely to stem from different memory access patterns between the real-world and the synthetic meshes. We attest to this likelihood by profiling parallel Poisson on Wolfdale and on Clovertown using OProfile [27]. Table 5 shows the results for Mesh15542876 and Mesh4091243 representing the synthetic meshes and NLR the real-world mesh. Clearly, the larger $t_i^{\text{oper}}$
Figure 3: Estimating $t_{oper}$ on Wolfdale and Clovertown processors

Table 5: Memory access behavior of Poisson using different test meshes

<table>
<thead>
<tr>
<th></th>
<th>Wolfdale</th>
<th></th>
<th>Clovertown</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mesh15542876</td>
<td>NLR</td>
<td>Mesh4091243</td>
<td>NLR</td>
</tr>
<tr>
<td>Subdomain size per core, $s$</td>
<td>1,942.860</td>
<td>2,040.107</td>
<td>510.873</td>
<td>520.436</td>
</tr>
<tr>
<td>Subdomain size per core (on log scale), Log($s$)</td>
<td>6.288</td>
<td>6.310</td>
<td>5.708</td>
<td>5.716</td>
</tr>
<tr>
<td>$t_{oper}$, millisecond</td>
<td>11.929</td>
<td>13.974</td>
<td>13.781</td>
<td>21.432</td>
</tr>
<tr>
<td>L2 cache miss ratio, %</td>
<td>38.897</td>
<td>42.423</td>
<td>22.781</td>
<td>21.432</td>
</tr>
<tr>
<td>Total bus bandwidth, MB/s</td>
<td>219.313</td>
<td>711.470</td>
<td>215.097</td>
<td>250.885</td>
</tr>
</tbody>
</table>

for NLR compared with the synthetic meshes is attributed to less efficient memory layout as evidenced by higher cache miss rate and bus bandwidth consumed, albeit equal workload.

4.2.3 Point-to-Point Communication

Intra-node contention can lead to significant difference in communication performance of the memory hierarchy. Unfortunately, a simple pingpong benchmark is insufficient to capture the contention effect on multi-core systems. Hence, a contention-aware methodology is of crucial importance to ensure that the extracted parameter values are accurate to reflect the degraded performance during parallel execution.

We used the WICON benchmark [30] that quantifies message latencies in the presence of network contention. The bisection communications adopted by WICON, however, do not suffice to represent the contention behavior of FEM applications. For execution on $P$ processors, WICON creates $P/2$ pairs of concurrent point-to-point communications. Nevertheless, a typical FEM application has far more communication pairs than $P/2$ because each process can involve in boundary exchange with more than one neighbor. Furthermore, WICON does not consider the effect of memory contention on communication performance. In a parallel implementation of FEM applications, message transmission is often overlapped with computation for improved performance. This overlap reduces the effective bandwidth as memory copies occur simultaneously with computation and message transfer.

To include memory contention in our benchmarking activity, we

1. increased the number of communication pairs (participating processes, $-np$) based on the commu-
Table 6: Maximum Communication Times of Poisson on the Clovertown node

<table>
<thead>
<tr>
<th>Mesh</th>
<th>Measurement (millsec/iteration)</th>
<th>Estimation (millsec/iteration)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>c = 2</td>
</tr>
<tr>
<td>NACA0015</td>
<td>2.724</td>
<td>104</td>
</tr>
<tr>
<td>M6</td>
<td>35.353</td>
<td>177</td>
</tr>
<tr>
<td>AS365</td>
<td>23.026</td>
<td>164</td>
</tr>
<tr>
<td>NLR</td>
<td>59.705</td>
<td>201</td>
</tr>
</tbody>
</table>

Fig. 4: Estimating (a) intra-node latencies on Clovertown, and (b) inter-node latencies on Wolfdale Cluster

Table 6 compares maximum communication times of running 8 processes (-np = 8) with estimated times derived from the fitting. As seen in the table, no single c value is able to predict the communication times of all test instances; NACA0015 is best estimated when using c = 128 whereas NLR c = 512. Nevertheless, the approach we used to find the c values for model validation hardly scales with processor count and the number of communication levels on the topology. An ideal case is that the application is examined beforehand such that c can be determined based on some profiled information.

1. modified WICON such that the memory copies happen at the same time as communication, i.e., each of halves of the ranks performs memory copies from an array sized c × message size, where c is a constant multiplier to allow proportional growth of memory access with communication volume, and

2. modified WICON such that the memory copies happen at the same time as communication, i.e., each of halves of the ranks performs memory copies from an array sized c × message size, where c is a constant multiplier to allow proportional growth of memory access with communication volume, and

3. modified the process mapping in WICON such that only intra- or inter-node communication takes place when profiling the performance of different levels.

Fig. 4(a) plots the intra-node message latencies as a function of message size for -np = 48 and c = 2, 128, and 512 while running the modified WICON on the 8-core Clovertown node. Fig. 4(b) shows the benchmark results of the inter-node communication on Wolfdale Cluster by increasing -np from 2 to 8. Bullets are measured data and dashed lines are the fit to the data. For brevity, results of other values of -np and c are omitted. We fit the data to obtain the best-approximated $t_{Q2}^i$ and $t_{Q3}^i$. Table 6 compares maximum communication times of running 8 processes (-np = 8) with estimated times derived from the fitting. As seen in the table, no single c value is able to predict the communication times of all test instances; NACA0015 is best estimated when using c = 128 whereas NLR c = 512. Nevertheless, the approach we used to find the c values for model validation hardly scales with processor count and the number of communication levels on the topology. An ideal case is that the application is examined beforehand such that c can be determined based on some profiled information.

6We implemented the non-contiguous data-access pattern of type fixed-length block with fixed stride [31].
4.3 Model Validation

With the empirical values of parameters attained in Sect. 4.2, we validate our performance model by comparing Poisson’s actual computation, communication, and execution times with estimation in homogeneous and heterogeneous environments. Note that the actual execution times are of non-blocking communication whereas the estimated runtime is calculated based on Eq. 10. For the homogeneous case, we used gpmetis 5.0 [21] to partition the test instances for Wolfdale Cluster and on Clovertown Cluster. For the heterogeneous case, we modeled the test bed as a three-level hypercluster topology and used our R-HierF partitioning framework [20] together with JOSTLE 3.1 [32] to create variable-sized subdomains. To estimate total saving time, we let $T_{\text{saving}}(N)$ be the difference in maximum communication times of all processors between blocking and non-blocking operations. The blocking times are measured by replacing the non-blocking MPI primitives, i.e.,

```c
foreach iteration {
    // Start exchanging boundary values with all neighbors
    foreach neighbor {
        MPI_Irecv()
        MPI_Isend()
    }
    // Computation on local data
    ...
    // Wait until all remote data are received
    MPI_Waitall()
    // Computation on remote data
    ...
}
```

with the blocking MPI_Sendrecv(), i.e.,

```c
foreach iteration {
    // Computation on local data
    ...
    // Exchange boundary values with all neighbors
    foreach neighbor {
        MPI_Sendrecv()
    }
    // Computation on remote data
    ...
}
```

Fig. 5 compares maximum computation times over homogeneous processors whilst Fig. 6 reports the evaluation as a function of load fraction $f$ assigned to Wolfdale Cluster in the heterogeneous hypercluster environment. Overall the estimation error spans -7.3 to 1.9 percent for the homogeneous clusters and -20.8 to 17.2 percent for the heterogeneous hypercluster. Omitting the -20.8 percent outlier of the AS365 mesh at $f = 0.7$, the error ranges between -14.1 and 17.2 percent. We believe that this outlier is due to a flaw in AS365 which causes disconnected vertices during the partitioning process. Also notice that the computation times of all test instances are the minimum for $f = 0.3$, suggesting load balance on the two clusters compared with other assignment ratios.

Table 7 and 8 give the estimated maximum communication times over homogeneous processors on Wolfdale Cluster and Clovertown Cluster respectively. The results are obtained based on the benchmarking methodology mentioned in Sect. 4.2.3. Hence, the actual times measured here are of blocking
communications. Numbers in bold signify the estimation errors which are less than 10 percent when using the corresponding \( c \) and \(-\text{np}\) values.

Table 9 compares the estimation with actual runtimes on Wolfdale Cluster and Clovertown Cluster respectively. The measured \( T_{\text{saving}}(N) \) values as well as the estimation without considering overlap, i.e., letting \( T_{\text{saving}}(N) = 0 \), are also displayed for assessment. When the effect of overlap is taken into account, the NLR mesh reaps the greatest benefit with the prediction error reduces from 54.41 to 1.49 percent while running on Clovertown Cluster and from 12.82 to 5.44 percent on Wolfdale Cluster. Surprisingly, the runtimes of M6 and AS365 on Clovertown Cluster are still over-estimated more than 20 percent. The misprediction is due likely to insignificant gain in using non-blocking communication, with the saving times less than 5 milliseconds per iteration compared with NLR of almost 50. MPE logging \[33\] confirms that a process of M6 incurs long waiting time (the Waitall event of rank 11 circled by a dotted line in Fig. 7 top) whereas NLR does not suffer from the impact of this cost (rank 6 and 7 in a dotted cirlce of Fig. 7 bottom). We posit that the asymmetric interaction between communication and computation \[34\] causes memory or network interface contention on one of the multicores, thereby leading to unnecessary waiting time albeit equal-sized subdomains running on symmetric hardware architecture.

5 Discussion

Prediction inaccuracies can occur from oversimplifications in the model \[5\] and the benchmarking methodology \[35\]. Assumptions such as blocking communication, minimal system contention, and non-overlapping between communications at the levels of memory hierarchy are often made to reduce complexity. Nevertheless, factors that seemed unlikely to affect performance analysis in a simple topology might cause
unreasonable errors in a multi-level multi-core environment. We have demonstrated the consequences of neglecting contention in measuring single-processor performance under concurrent computation (see Sect. 4.2.2). While we strive to minimize inaccuracies by relaxing the assumptions, the misprediction in estimating model parameters still persists. For instance, measuring precise saving in communication times is challenging unless the application code is carefully profiled to identify correlation between various events. Detailed exploration is required to verify that this aspect can indeed be excluded without sacrificing overall accuracy.

As explained in Sect. 4.2.2, the estimation of $t_{\text{comm}}$ relies on various factors including the memory reference pattern of the application. We conjecture that memory access behavior is influenced by the properties of the input meshes. To illustrate, Fig. 8 displays the AS365 mesh and a synthetic mesh created from a square domain. Although the degree of connectivity, i.e., the ratio of number of edges and number of vertices, of both meshes are equal, what distinguishes them is the triangle density. Notice that AS365 possesses smaller triangular elements around the boundary of the object of interest to simulate the physics on certain regions. In contrast, the synthetic mesh created with a fixed area constraint features no element with area greater than the prescribed constraint. Another property that is likely to make a difference is the sparsity pattern. To explore how sparsity pattern affects memory access pattern, we modified the NLR mesh by shuffling its nodes and elements arbitrarily. While we observed no significant changes in performance, studying the mesh characteristics may give important insights into optimizing the partitioning refinement.

In Sect. 4.2.3, we modified the WICON benchmark to include intra-node memory contention during message transmission. The adaptation raises a fundamental question, i.e., how much memory access to
Table 8: Maximum Communication Times of Poisson over all processors on Clovertown Cluster

<table>
<thead>
<tr>
<th>Mesh</th>
<th>Measurement (millisec/iteration)</th>
<th>Estimation (millisec/iteration)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NACA0015</td>
<td>c = 128</td>
<td>c = 448</td>
</tr>
<tr>
<td></td>
<td>1.796</td>
<td>1.474</td>
</tr>
<tr>
<td>M6</td>
<td>21.933</td>
<td>3.620</td>
</tr>
<tr>
<td>AS365</td>
<td>21.414</td>
<td>3.586</td>
</tr>
<tr>
<td>NLR</td>
<td>49.705</td>
<td>3.853</td>
</tr>
</tbody>
</table>

Table 9: Execution times of Poisson on Wolfdale Cluster and Clovertown Cluster

<table>
<thead>
<tr>
<th>Mesh</th>
<th>Measurement (millisec/iteration)</th>
<th>T_estimation(N)</th>
<th>Estimation (millisec/iteration)</th>
<th>Estimation Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>no overlap</td>
<td>with overlap</td>
<td>no overlap</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Wofldale Cluster:</td>
<td>Clovertown Cluster:</td>
<td></td>
</tr>
<tr>
<td>NACA0015</td>
<td>38.683</td>
<td>6.805</td>
<td>43.552</td>
<td>36.747</td>
</tr>
<tr>
<td>M6</td>
<td>218.047</td>
<td>12.404</td>
<td>229.952</td>
<td>217.548</td>
</tr>
<tr>
<td>AS365</td>
<td>238.495</td>
<td>5.218</td>
<td>243.743</td>
<td>238.525</td>
</tr>
<tr>
<td>NLR</td>
<td>272.812</td>
<td>20.144</td>
<td>307.785</td>
<td>287.642</td>
</tr>
</tbody>
</table>

add so that WICON reflects the memory communication costs of the application? In other words, how could we identify the values of \( c \) without resorting to actual measurements? Unfortunately, we found no global value which suits all cases because memory contention can vary over resource properties, workload characteristics, mapping options, and the hierarchical structure of interconnects. Even for the same graph, the memory access behavior will change substantially with different number of communication levels in the topology. An area of future work is to quantify memory contention, decide the \( c \) value adaptively, and use this information to benchmark the communication performance in a way that mimics application behavior as closely as possible.

There exist two ways to model overlap, i.e., either directly in the communication modeling or separately from communication and computation. The latter approach is known as the fundamental equation of modeling [5] which first models computation and communication as non-overlapping and then deducts the total costs by the saving time. We believe that this approach is less complicated than direct modeling. Given that general-purpose benchmark programs measure the performance in a non-overlap condition, it appears impossible to estimate only the communication time not overlapped with computation. Further, this approach is convenient for a hypercluster topology because the overlap is expressed as an aggregated saving irrespective of which form the overlap is. The drawback, of course, is that we are unable to investigate the effect of neglecting certain kind of overlap, e.g., communication overlap at different levels of the memory hierarchy. Researchers have begun to acknowledge the importance of considering overlap. However, experimentation about how to acquire the value on real systems is still limited in the literature. Future work may include profiling the code to explore ways to model overlap and embrace the notion into performance modeling.
Figure 7: Jumpshot [33] showing event traces of the M6 mesh (top) and the NLR mesh (bottom). M6 suffers from long waiting time whereas NLR does not.

6 Conclusions

In this paper we have described a performance model for a three-level hierarchical decomposition of an FEM application on heterogeneous multi-core clusters. We validated the model using real-world finite-element meshes and our parallel Poisson solver. With the assumptions that (i) the communication is non-blocking, (ii) there exists overlaps of communication-computation and communications at different levels, and (iii) the impact of resource contention is non-negligible, our model predicts execution times on homogeneous clusters with a maximum error of 5.4 percent across 6 out of 8 cases and computation times on a heterogeneous hypercluster with a worst-case error of 17.2 percent across all data sets.

The developed model can be used to quantify the impact of resource heterogeneity and contention on graph partitioning. We plan to incorporate the model into our R-HierLB framework to refine the partition without actual execution on target machines. Tentatively, the estimated runtime serves as input to a line search algorithm to seek the next local minima (lower runtime). The iterative process stops when the difference between the current and previous estimated time is less than a pre-defined threshold. The enhanced framework eases the burden of developers as optimization based on application behavior can
Figure 8: The AS365 mesh (left), and a synthetic mesh created from a square domain (right)

now be conducted offline and results are reproducible without being perturbed by system loads.

Acknowledgments

Experimental testbed was funded by Ministry of Science, Technology and Innovation of Malaysia under eScienceFund. The authors would like to thank Jian Tao Zhang for creating test instances and anonymous reviewers for helpful comments.

References


