# Infrastructure for In Situ System Monitoring and Application Data Analysis

Jim Brandt, Karen Devine, and Ann Gentile
Sandia National Laboratories*
Albuquerque, NM, USA
(brandt|kddevin|gentile)@sandia.gov

## ABSTRACT

We present an architecture for high-performance computers that integrates in situ analysis of hardware and system monitoring data with application-specific data to reduce application runtimes and improve overall platform utilization. Large-scale high-performance computing systems typically use monitoring as a tool unrelated to application execution. Monitoring data flows from sampling points to a centralized off-system machine for storage and post-processing when root-cause analysis is required. Along the way, it may also be used for instantaneous threshold-based error detection. Applications can know their application state and possibly allocated resource state, but typically, they have no insight into globally shared resource state that may affect their execution. By analyzing performance data in situ rather than off-line, we enable applications to make real-time decisions about their resource utilization. We address the particular case of in situ network congestion analysis and its potential to improve task placement and data partitioning. We present several design and analysis considerations.

## 1. INTRODUCTION

In high-performance computing (HPC) systems, the application execution environment is usually completely separated from system monitoring capabilities. While applications may access some node-level utilization data from user space (e.g., cpu, memory utilization), there is additional performance data that is not accessible from user space, particularly about shared resources (e.g., network load, shared file system load, burst buffer use), that could be used to better enable application-to-resource mapping. System monitoring capabilities exist that run at privileged levels and thus have access to global and restricted data. However, such systems are typically used only for instantaneous detection of critical conditions (e.g., high temperatures, high loads) or to write raw data to an off-system store for post-processing analysis. In addition, typical monitoring system overheads severely limit the data collection frequencies and, hence, the useability of their data to influence applications.

Tight integration of system monitoring information with an application's execution environment could provide significant opportunities to improve an application's execution performance. Such integration requires a) lightweight, appropriate-frequency monitoring data, b) data analysis within the monitoring framework to provide run-time information with low latency, c) interfaces by which an application or system service could quickly access such data, and d) additional in situ analyses and responses that include the application state information. Unlike other infrastructures which address in situ analysis of scientific data, this infrastructure exposes system state data for analysis both within the monitoring system and to the running application. The monitoring data analysis step can also be used for greater insight for system adminstrators without a need for increased data handling, as streaming analysis can enable more intelligent, and thus reduced, short-term data storage.

In this paper, we present an infrastructure that enables this type of integration and supports general analysis in the overall workflow. Ultimately, we anticipate this infrastructure will provide a system service available to applications as well as to schedulers and resource managers. Use of this service will improve execution and system performance via improved application-to-resource mapping and/or through better scheduling and resource allocations.

In Section 2, we describe an infrastructure that more tightly couples the monitoring system with the application execution environment, enabling in situ analysis of monitoring system data and application data within an execution workflow. In Section 3, we present requirements, details, and advantages of the infrastructure applied to in situ network contention analysis and dynamic task placement. In Section 4, we present our current state of implementation, remaining work, and future plans.

## 2. INFRASTRUCTURE

We describe our monitoring and analysis infrastructure and how it enables in situ analysis of monitoring and application data for use by applications and/or system services. The basic capabilities of the infrastructure are 1) data collection and aggregation, 2) monitoring data analysis, 3) application analysis, and 4) response triggering (feedback). A

*Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000



*ISAV2015 November 15-20, 2015, Austin, TX, USA*



DOI: http://dx.doi.org/10.1145/2828612.2828621

high-level diagram of the architecture is shown in Figure 1. The components are described in greater detail below.

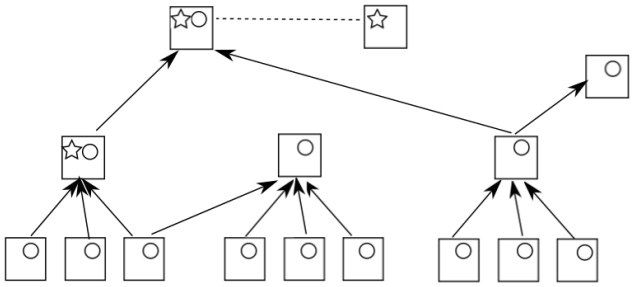

**Figure 1: Architecture overview. Nodes (and other data sources) are squares. Circles are monitoring system daemons which support data collection and storage/analysis plugins. Data aggregation nodes collect data; arrows show direction of data flow. Arbitrary topologies are supported; thus analysis can occur on any subset of data. Application and other analysis consumers are stars and are typically not globally distributed. Analysis consumers that are co-located with the monitoring system daemons can interact directly with them; remote access to the monitoring system daemons is also supported.**

### *System Data Collection, Aggregation, and Analysis.*

In our architecture, we leverage the Lightweight Distributed Metric Service (LDMS) for data collection and aggregation [2]. In the LDMS framework, daemons run on the nodes, with plugins for data sampling and storage. Daemons can also aggregate data from other LDMS daemons in arbitrary topologies. Thus, multiple aggregation points can be configured to collect data from disjoint and/or overlapping sources.

Because of the low overhead of LDMS, data collection intervals on production systems typically range from seconds to a minute. A recent enhancement to LDMS is a vector data type which enables more frequent sampling without the need for a corresponding increase in aggregation frequency. The vector type thus makes available even higher frequency data.

The design of our infrastructure enables us to perform full resolution data analysis directly on the data as it is streaming through the aggregators, thus avoiding the overhead of first storing to disk and then reading back for subsequent analysis. To do this, we take advantage of the fact that storage plugins have direct access to the in-memory data of LDMS aggregators. Thus, we have implemented analysis plugins, based on the storage plugin, that take advantage of this access to perform analyses directly on this data. The most recent analysis and limited historical data (e.g, last defined number of data points) are then kept in-memory in the plugin; the complete data can additionally still be stored or discarded. By integrating the analysis with the data aggregator and performing continuous analysis, the results are more immediately available whenever they are required. This integration also enables more sophisticated analysis, including consideration of variable frequency time history of the system state, rather than an analysis based on only the most recent data. By splitting the data among the aggregators, analyses can be performed in parallel on subsets of the data, and the results combined, if desired, by the ultimate data consumer.

### *Monitoring Analysis Interface.*

Low latency is a requirement for applications and system services to derive maximum benefit from in situ analysis of run-time monitoring data in their computations and responses. If acquisition of pertinent data increases the runtime, its use becomes detremental. Thus, the design includes the option for the data consumer to register, at the outset, its intent to gather a defined set of information from the analysis plugin. For example, the registered information could be a set of routes for which network congestion state values are desired. The required calculations could then be included as part of the aggregator data collection and the query latency could be as short as a round trip communication plus data transmission time. The latency could be further reduced if the information were made available to appropriate application elements via memory mapped regions on their local hosts.

The analysis and shared data architecture are shown in Figure 2. This figure includes data operations in the plugin and also the memory mapped region for sharing data between the plugin and the data consumer.

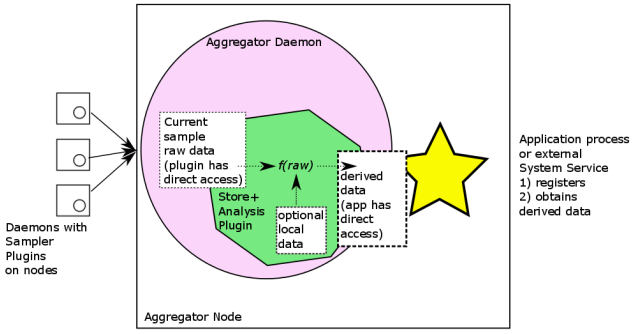


**Figure 2: Plugin and shared in-memory data architecture supporting both in situ analysis in the monitoring system and in the application/system service. The analysis plugin can optionally function as a storage plugin and store the raw and/or derived data.**

## 3. USE CASE: NETWORK CONGESTION AND APPLICATION RESPONSE

Application performance can be affected by a number of factors: load imbalance, data locality, communication overhead, etc. Some of these factors can be controlled by careful algorithmic design. But some are impacted by the state of the computing system at execution time. For example, network contention with other applications running concurrently can slow an application's communication in unpredictable ways. Conversely, suboptimal placement of an application's tasks on nodes can increase the total network bandwidth used, impacting other running jobs. Even the particular choice of nodes assigned to an application can impact its performance, as messages may have to travel many hops between allocated nodes. Using dynamic performance data and analysis to make decisions about an application's use of resources can alleviate some of this unpredicability.

Use of system performance data in applications places re-

quirements on the data collection and analysis infrastructure. Performance data needs to be relevant to avoid making decisions based on stale information; in-situ aggregation in analysis plug-ins allows data to be provided in real-time. However, to prevent applications from thrashing between decisions, data also should not be too jittery. Analysis must provide reasonable expectations of future system performance based on recent past performance. Off-line *a posteriori* analysis can help identify patterns and trends in system-level usage; for example, it can provide probability distributions that, combined with real-time data, help predict near-term system performance based on current state. These distributions can then serve as initial conditions for in-situ evaluation of current and recent state. The in-situ analysis must be fast, so that potential benefits of redistributing data are not outweighted by the cost of analysis.

In the following sections, we present one use case for our infrastructure: analysis of network congestion and placement of MPI tasks on cores to avoid detected congestion.

## 3.1 Network Congestion Analysis

We demonstrate benefits of our infrastructure with experiments on Cray XE systems with Gemini [4] networks. The Gemini network is a 3D torus. Each Gemini routing element connects directly to two nodes. Traffic between any source and destination node is sent via a deterministic multi-hop route. (There are 6 directional options at any intermediate router,`(X/Y/Z) (+/-)`, and traffic in each of the 6 directions is independent.) Congestion on intermediate and shared hops can affect the application's performance; however, the nodes directly connected to the Gemini routers for the intermediate hops may not be part of the application's allocation and thus information on network state of those hops would not be accessible by the application. A global monitoring system analysis can provide information about network contention that can then be used with application communication patterns to enable allocation and mapping decisions. Details of the route identification, calculations of a) maximum percentage of bandwidth used and b) maximum percent time spent in stalls among all hops along a route, and the use of these calculations as a measurement of congestion are given in [9]. In that work, calculations were performed on-demand.

The ability to analyze data at run-time in the monitoring system aggregator enables us to perform *continuous* analyses of any type on any subsets of data that we wish. The most comprehensive analysis would require data from all nodes, in conjunction with the static routing map, which would require the analysis to be performed at a global system aggregator. While the overall computation could be significant (e.g., 27648*27648 unique route calculations for the NCSA's BlueWaters [1]), there are a number of possible optimizations. Registration of the routes of interest is the most significant reduction for computations required. Further, we perform all routes' calculations in parallel. For analysis algorithms that can be further subdivided, such as the maximum value of a quantity for a hop along a route, we can do the analysis across multiple aggregators each with only a subset of the data and then combine the results. Finally, examination of the data may provide some reduction of the analysis. For instance, our preliminary examinations of data from BlueWaters have shown significantly less congestion in the `Z+/-` hops as compared to the others, so equivalently effective calculations might be done without consideration of these values.

An advantage of continuous analysis is that detection of significant change could be used to trigger application reconfiguration. For example, a newly launched communication-intensive job may create long-lived, significant congestion; discovery of such an event could be used to advise other jobs to redistribute their tasks or data. An indication of `X+` link-congestion duration from one arbitrarily selected (not representative) day on BlueWaters is shown in Figure 3. We show $P(t+dt)$, the probability that if the percent time spent in credit stalls on a given link is over some value at time $t$, that it still will be over that value at time $t+dt$. Results for four different values are shown in the figure. (Data was collected at one-minute intervals [2] for each router in each of the 6 directions). Although congested links were infrequent (values of percent time spent in credit stalls $\geq 30\%$ constituted $\sim 0.6\%$ of the data values of this entire day), when higher congestion values occured they exhibited significant durations of more than 20 minutes (e.g, a link with value $\geq 50\%$ has over 60% probability of having a value $\geq 50\%$ 20 minutes later (blue line)). The congested links would thus affect all applications communicating via routes which contain these links. Characterizations such as this can also be used to guide the data collection/recomputation frequency.

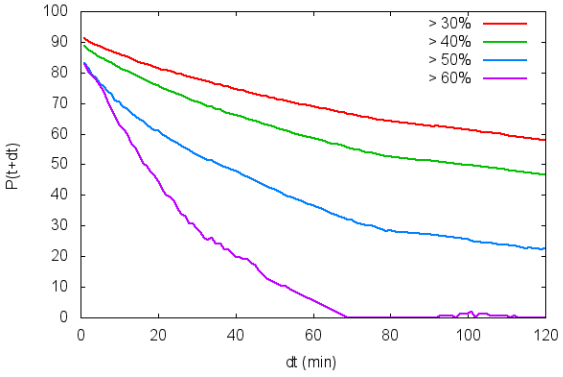


**Figure 3: $P(t+dt)$ vs $dt$: Probability that if the percentage of time spent in credit stalls on an `X+` link is greater than some value at time $t$, it will also be greater than that value at time $t+dt$. $P(t+dt)$ is shown for 4 different values. Data was taken on one arbitrary day on a large-scale Cray XE system at one minute intervals.**

Our infrastructure allows applications to perform more relevant interpretations of hardware metrics than would be possible by directly querying hardware counters (even beyond the issue of off-node data accessibility). For example, the source of the Gemini network data (`gpcdr` [11]) uses cached data values that are not updated until a specified time window has passed. Cray's initial implementation of this source did not check for negative time values. A negative clock reset results in the aged, and now inaccurate, values being reported until the clock catches up with the time of the reset. This condition is easily discoverable at run-time by the monitoring system by detection of gross timestamp discrepencies and/or changes across large sets of collected data. Generation of a mitigating response would be trivial from a monitoring-system perspective.

Additionally, on Cray XE/XK systems there is a mechanism that quiesces the network (stopping injections into the network) in response to extreme congestion or routing around failed links. This situation causes contention metric values to drop, which may look like congestion has abated, when, in reality, the network may be overloaded. Indication of quiesces and rerouting are privileged and thus more easily obtained from a monitoring system rather than from a user-space entity. Inclusion of quiesce events into the analysis, in this case, would prevent erroneous reporting of low congestion values and provide new path computations to accommodate re-routing around failures.

## 3.2 Congestion-Aware MPI Task Placement

The placement of MPI tasks in cores has been shown to impact the scalability of even simple stencil-based applications (e.g., [8, 3, 6]). Typically, job schedulers assign MPI tasks in a linear or random fashion to the nodes allocated to the jobs. These default placements of MPI tasks in cores do not account for interdependence among the MPI tasks, the static topology of the network, or the real-time state of the computer. Several algorithms exist to account for the static network topology and task interdependence [18, 7, 15, 12]. These algorithms take as input geometric or graph-based representations of both the application data and the network topology, and map one representation onto the other.

In [9], we investigated the placement of MPI tasks onto allocated cores to avoid real-time network congestion. As with most graph-based static mapping tools, our dynamic placement used two graphs as input. One graph described the communication patterns of the underlying application. Graph vertices represented MPI tasks and weighted edges represented the volume of communication (number of bytes) to be sent between pairs of tasks. This graph was provided by the application. A second graph described the cost of communication between the nodes allocated to the application. Graph vertices represented the nodes, and weighted graph edges represented the cost to communicate between pairs of nodes. For static mapping, these weights could be, e.g., the number of network hops between nodes. But for our dynamic placement, the edge weights were obtained from a separate software component (the ResourceOracle) which queried LDMS for raw data and calculated the percentage of used bandwidth or time spent in credit stalls along the route between a pair of nodes. Routes with greater congestion used higher edge weights in the architecture graph, while routes with little congestion had lower weights. The two graphs were then passed to the Scotch [18] graph partitioning and mapping toolkit which, through recursive bipartitioning of the two graphs, assigned interdependent MPI tasks to nodes with low communication cost between them. We showed that we could recover up to 49% of the execution time lost to congestion in small-scale experiments.

Our improved infrastructure overcomes several shortcomings in this prior work to enable us to apply our approach to larger-scale systems and with more responsive analysis. In the prior work, the network congestion analysis was performed on-demand and required an explict query to the ResourceOracle and thus the aggregator by a remapping analysis component. That data was obtained over a socket with a well-defined message format for the query and the response. With our new infrastructure, the integration of the network analysis with LDMS via the plugin reduces the overhead and latency to obtain the network contention data and do the route congestion analysis. The plugin performs continuous analysis which enables us to discover changing contention in the routes of interests and pro-actively trigger remapping of tasks. Moreover, the memory mapped region reduces the latency for obtaining the congestion data to be used in the mapping analysis as compared to the socket-based approach.

## 3.3 Congestion-Aware Data Partitioning

Many applications (e.g., adaptive finite element methods [5, 13, 17, 23, 25]; particle-in-cell methods [19, 20, 26]) use dynamic data partitioning to balance changing workloads and/or maintain locality during a simulation. Such applications could benefit further by including real-time system performance information in their repartitioning. In traditional graph-based partitioners [22, 10, 14, 16], application data is modeled as a graph: weighted vertices represent data objects and their computational cost, while weighted edges represent the strength of dependence between data objects. The graph is then divided into a specified number of equally weighted parts so that the weight of edges cut by part boundaries is small. A few graph partitioners [18, 24] also accept a static graph of the network to try to align application communication across part boundaries that span low-cost communication links in the network graph. Replacing the static network graph with a graph weighted with dynamic information about congestion or used bandwidth holds the potential to make partitioning strategies more effective by allowing them to avoid network bottlenecks. Continuous analysis detecting changes in network performance can also trigger repartitioning for the application.

# 4. CURRENT STATE AND FUTURE WORK

We have presented an infrastructure for analyzing real-time system monitoring data and providing it to applications for decisions about task placement and work distribution. Currently, we have implemented the continuous analysis plugin with the parallel network analysis described here. We have not yet implemented the memory mapped region for more tightly coupled sharing of analysis results with applications; we still provide a socket interface to that data.

We are investigating direct inclusion of network quiesce event data and detection of significant decrease of network congestion values on all links as indicators of quiesce events. We are also performing experiments to characterize the impact of contention measures on application performance, to refine the weighting of the machine graph for remapping.

We addressed Cray's Gemini network, because it has deterministic routing. However, we are also investigating congestion measures on Cray's Aries-based dragonfly network. The benefits to performance of the Aries adaptive routing are not clear and detailed information from the Aries performance counters could be used for better understanding of the existing routing algorithms, exploration of other algorithms, and enabling finer grained response to congestion.

While the work and use cases presented have focused on application use of network congestion information, there is a wealth of other information such as power consumption, thermal information, storage bandwidth use, that could be exposed via this same interface for use by both applications and other system entities such as schedulers, resource managers, analysis tools, etc. For example, memory contention has been used to manage concurrency in multithreaded ap-

plications [21], and CPU utilization could be used in setting appropriate part sizes in dynamic load balancing tools.

Batch schedulers currently interact with resource managers to assign nodes to jobs based only on availability and expected job duration. If these system services also had access to current and expected network congestion information within the system, they could place jobs away from heavily congested regions, reducing the jobs' communication time and increasing overall system throughput. Likewise having access to current shared filesystem bandwidth headroom in conjunction with historic application use could inform these system services to enable scheduling and resource allocation that could minimize contention and increase system throughput. A small amount of application information, historic and/or run-time, (e.g., whether the job is communication-bound or computation-bound) would allow more customized node allocations that would avoid or tolerate congested regions accordingly.

# 5. REFERENCES


[1] Blue Waters. https://bluewaters.ncsa.illinois.edu.

[2] A. Agelastos, B. Allan, J. Brandt, P. Cassella, J. Enos, J. Fullop, A. Gentile, S. Monk, N. Naksinehaboon, J. Ogden, M. Rajan, M. Showerman, J. Stevenson, N. Taerat, and T. Tucker. Lightweight Distributed Metric Service: A scalable infrastructure for continuous monitoring of large scale computing systems and applications. In *Proc. Int'l Conf. for High Perf. Storage, Networking, & Analysis (SC14)*, 2014.

[3] H. M. Aktulga, C. Yang, E. G. Ng, P. Maris, and J. P. Vary. Topology-aware mappings for large-scale eigenvalue problems. In *Euro-Par 2012 Parallel Processing*, pages 830–842. Springer, 2012.

[4] R. Alverson, D. Roweth, and L. Kaplan. The Gemini System Interconnect. In *Proc. 2010 IEEE 18th Annual Symp. High Perf. Interconnects (HOTI)*, 2010.

[5] S. T. Barnard and H. Simon. A parallel implementation of multilevel recursive spectral bisection for application to adaptive unstructured meshes. In *Proc. 7th SIAM Conf. Parallel Proc. Scientific Computing*, pages 627–632, 1995.

[6] R. Barrett, C. Vaughan, S. Hammond, and D. Roweth. Reducing the bulk of the bulk synchronous parallel model. *Parallel Process Lett*, 23(4), 2013.

[7] A. Bhatelé and L. V. Kalé. Benefits of Topology Aware Mapping for Mesh Interconnects. *Parallel Processing Letters (Special issue on Large-Scale Parallel Processing)*, 18(4):549–566, 2008.

[8] A. Bhatele, L. V. Kale, and S. Kumar. Dynamic topology aware load balancing algorithms for molecular dynamics applications. In *Proc 23rd Intl Conf Supercomputing*, pages 110–116. ACM, 2009.

[9] J. Brandt, K. Devine, A. Gentile, and K. Pedretti. Demonstrating improved application performance using dynamic monitoring and task mapping. In *Wksp Monitoring & Analysis for High Perf. Comput. Sys. Plus Applications (HPCMASPA) Proc. IEEE Intl. Conf. Cluster Comput. (CLUSTER)*, 2014.

[10] T. Bui and C. Jones. A heuristic for reducing fill in sparse matrix factorization. In *Proc. 6th SIAM Conf. Parallel Processing for Scientific Computing*, pages 445–452. SIAM, 1993.

[11] Cray Inc. Managing System Software for the Cray Linux Environment. Technical Report Cray Doc S-2393-4202, 2013.

[12] M. Deveci, S. Rajamanickam, V. Leung, K. Pedretti, S. Olivier, D. Bunde, U. V. Catalyurek, and K. Devine. Exploiting geometric partitioning in task mapping for parallel computers. In *Proc. 28th Int'l IEEE Parallel & Distrib. Proc. Symp.*, 2014.

[13] K. Devine and J. Flaherty. Parallel adaptive *hp*-refinement techniques for conservation laws. *Appl. Numer. Math.*, 20:367–386, 1996.

[14] B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. In *Proc. Supercomputing '95*. ACM, December 1995.

[15] T. Hoefler and M. Snir. Generic topology mapping strategies for large-scale parallel architectures. In *Proc 25th Intl Conf Supercomputing*, pages 75–84. ACM, 2011.

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

[17] A. Patra and J. T. Oden. Problem decomposition for adaptive *hp* finite element methods. *J. Computing Systems in Engg.*, 6(2), 1995.

[18] F. Pellegrini and J. Roman. Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In *High-Performance Computing and Networking*, pages 493–498. Springer, 1996.

[19] J. R. Pilkington and S. B. Baden. Dynamic partitioning of non-uniform structured workloads with space-filling curves. *IEEE Trans. Parallel Distrib. Sys*, 7:288–300, 1996.

[20] S. Plimpton, S. Attaway, B. Hendrickson, J. Swegle, C. Vaughan, and D. Gardner. Transient dynamics simulations: Parallel algorithms for contact detection and smoothed particle hydrodynamics. *J. Parallel Distrib. Comput.*, 50:104–122, 1998.

[21] A. Porterfield, S. Olivier, S. Bhalachandra, and J. Prins. Power measurement and concurrency throttling for energy reduction in OpenMP programs. In *2013 IEEE 27th Intl. Parallel & Distrib. Proc. Symp. Workshops PhD Forum (IPDPSW)*, pages 1–8, May 2013.

[22] A. Pothen, H. Simon, and K. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Matrix Anal.*, 11(3):430–452, July 1990.

[23] K. Schloegel, G. Karypis, and V. Kumar. Multilevel diffusion algorithms for repartitioning of adaptive meshes. *J. Parallel Distrib. Comput.*, 47(2):109–124, 1997.

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

[25] C. Walshaw, M. Cross, and M. Everett. Parallel dynamic graph-partitioning for adaptive unstructured meshes. *J. Par. Dist. Comput.*, 47(2):102–108, 1997.

[26] M. S. Warren and J. K. Salmon. A parallel hashed oct-tree n-body algorithm. In *Proc. Supercomputing '93*, Portland, OR, Nov. 1993.