Super-Scalable Algorithms for Computing on 100,000 Processors *

Christian Engelmann and Al Geist

Computer Science and Mathematics Division,
Oak Ridge National Laboratory, Oak Ridge, TN 37831-6164, USA
{engelmannc,gst}@ornl.gov
http://www.csm.ornl.gov

Abstract. In the next five years, the number of processors in high-end systems for scientific computing is expected to rise to tens and even hundreds of thousands. For example, the IBM Blue Gene/L can have up to 128,000 processors and the delivery of the first system is scheduled for 2005. Existing deficiencies in scalability and fault-tolerance of scientific applications need to be addressed soon. If the number of processors grows by a magnitude and efficiency drops by a magnitude, the overall effective computing performance stays the same. Furthermore, the mean time to interrupt of high-end computer systems decreases with scale and complexity. In a 100,000-processor system, failures may occur every couple of minutes and traditional checkpointing may no longer be feasible. With this paper, we summarize our recent research in super-scalable algorithms for computing on 100,000 processors. We introduce the algorithm properties of scale invariance and natural fault tolerance, and discuss how they can be applied to two different classes of algorithms. We also describe a super-scalable diskless checkpointing algorithm for problems that can't be transformed into a super-scalable variant, or where other solutions are more efficient. Finally, a 100,000-processor simulator is presented as a platform for testing and experimentation.

1 Introduction

Today's top supercomputers are able to deliver several tens of TeraFLOPS of sustained performance for computational scientific research in areas like climate modeling, fusion energy and nanotechnology. If the steady increase in computing power stays on the track of Moore's Law, by 2010 the largest supercomputers in the world will be in the PetaFLOPS range. This trend is not solely based on improvements of individual processors, but also aided by ever-increasing parallelism. Currently, these systems scale for up to 10,000 processors. In the next five years the number of processors is expected to rise to tens and even hundreds of

* Research sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory (ORNL), managed by UT-Battelle, LLC for the U. S. Department of Energy under Contract No. DE-AC05-00OR22725.

thousands. For example, the IBM Blue Gene/L [1, 2] will have up to 128,000 low-powered processors shipped in densely populated compute nodes. The first Blue Gene/L system will be delivered in 2005. A prototype at IBM recently achieved a maximal LINPACK performance of 70 TeraFLOPS and currently holds the top spot in the Top 500 list of supercomputers.

Experiences with existing 10,000-processor machines show that the efficiency of scientific applications can be as low as 1%, which is equal to fully utilizing only 100 processors. Amdahl's Law shows how efficiency can drop off as the number of processors increases. If the number of processors grows by a magnitude and efficiency drops by a magnitude, the overall effective computing performance stays the same. Furthermore, the mean time to interrupt (MTTI) decreases with system scale and complexity. While reliability of individual components, such as network and storage, can be improved by redundancy, the number of system software issues increases due to complexity. Some of today's major supercomputing centers have already scheduled downtimes and unscheduled outages about every 40 hours. In a 100,000-processor machine, such system interrupts may occur as often as every couple of minutes. Network bottlenecks and latencies will make frequent coordinated checkpointing (once every hour) of applications, for fault-tolerance, almost impossible. Even with traditional checkpointing, it does not make sense to restart 99,999 processors because one failed! Finally, at some point the MTTI is going to exceed the time to restart.

In this paper, we summarize our recent research in super-scalable algorithms for high-end scientific computing on extreme-scale systems with 100,000 processors. First, we introduce the algorithmic properties of *scale invariance* and *natural fault tolerance*, and then we discuss how they can be applied to two different classes of algorithms. We also describe a super-scalable diskless checkpointing algorithm for problems that cannot be transformed into a super-scalable variant, or where other solutions are more efficient. We continue with a short description of our efforts in developing a 100,000-processor simulator as a platform for testing and experimentation. Finally, we close with a brief summary of the work and possible future directions.

2 Super-Scalable Algorithms

High-end computing on 100,000-processor systems requires fundamental rethinking of how algorithms can efficiently utilize such an enormous amount of processors. There are two major issues that need to be considered. The first is Amdahl's Law, and the need to reduce the serial fraction to a point where reasonable efficiency can be achieved. The second is the high probability of failures, and the need to survive in a way that does not involve global operations. In order to address these problems, we have established a foundation for a new class of algorithms called *super-scalable algorithms* [3] that have the properties of *scale invariance* and *natural fault tolerance*.

Scale invariance means that the individual tasks in a larger parallel job have a fixed maximum number of other tasks they communicate with, independent of

the total number in the application. For example, a finite difference algorithm has a constant number of neighbor tasks defined by its stencil, which is independent of the total number of tasks in the problem. Another example is a binary tree communication infrastructure, where each node is only connected to three other nodes. With scale invariance, individual tasks do not have to be concerned about failures throughout the system unless these failures happen to affect one of their neighbors. Conversely, dynamically adding replacement or additional tasks can be ignored by tasks not communicating with these new tasks.

However, scale invariance alone does not guarantee high efficiency of applications on 100,000-processor computing systems. The serial fraction of a parallel algorithm does not solely depend on the communication footprint, but also on hardware factors, such as I/O latencies and cache misses, that can quickly drive efficiency down even if the best-known algorithms are being used.

Scale invariance does not provide fault tolerance, but it enables isolation of the failure. However, most parallel algorithms designed today will deadlock, or worse, calculate the wrong answer, if one or more tasks fail. Fault tolerance needs to be handled locally by "self-healing" or natural fault tolerance.

Natural fault tolerance is the ability to tolerate failures through the mathematical properties of the algorithm itself, without requiring notification or recovery. It is not that the calculations are taken over by other tasks, but rather that the nature of the algorithm includes natural compensation for the lost information. For example, an iterative algorithm may require more iterations to converge, but it still converges despite lost information [4].

The maximum number of tasks that can fail, yet still obtain the correct answer, is problem dependent and still an open research question. We assume that the actual number of tasks lost during an application run will be a small fraction of the overall number of tasks. We based our research on the assumption that up to 100 out of 100,000 tasks may fail, which is only 0.1%. However, the time-to-solution increases dramatically when using traditional checkpointing or message logging schemes due to the large amount of processors involved and the centralized nature of existing solutions. We discuss a peer-to-peer diskless checkpointing alternative later in this paper.

Scale invariance and natural fault tolerance are rather restrictive requirements on algorithms, and when we began our research it was not clear that anything other than the most trivial applications, using the bag-of-tasks programming paradigm, would be able to meet these definitions. Such applications are (to a certain extend) scale invariant, because each task communicates only to send back its answer. They have fault tolerance, because tasks are farmed out and can be easily replaced. Task farming with on-the-fly fault tolerance by task replacement is a widely used technique today. Examples are SETI@HOME [5] and Condor [6, 7].

In the following sections, we will describe solutions for two different non-trivial classes of super-scalable algorithms. The first is where the problem can be formulated as some function of a local volume, such as for finite difference and finite element applications. The second is where the problem requires global

information, like in global minimum or maximum searches, that are often used to determine if an iterative algorithm has converged.

2.1 Local Information

Parallel applications where individual tasks only require information from a local region include finite difference and finite element solutions to differential equations. We combined two ideas, chaotic relaxation [8, 9] and meshless methods [10], to demonstrate that both super-scalable algorithm requirements, scale invariance and natural fault tolerance, can be achieved.

In a meshless finite difference algorithm with chaotic relaxation, each data point in the solution space is assigned to an independent task that asynchronously receives update messages from its neighbors, calculates its own value and sends update messages back to its neighbors. The programming model is similar to active messaging [11], but could be coded using PVM [12] or MPI [13].

We use a coordinate in a virtual space to identify each task. This virtual space may coincide with the solution space, for example in a 2-D Poisson problem. Based on the coordinate, we can form nearest neighbor as well as random peer-to-peer networks to experiment with the algorithm. Each message contains the sender coordinates, so that necessary metrics, such as distance, can be calculated at runtime. Update messages additionally contain the value of the sending task. The update processing routine reflects the mathematical definition of the task and its relation to its neighboring tasks, which in the case of the 2-D Poisson problem is an average of the surrounding values with a distance bias.

Early investigations in the 1970's showed that chaotic relaxation has quite restrictive convergence properties, which is the main reason why it never became popular. However, for 100,000-processor systems it may be time to once again look at this iteration-free method. When failures and failure recovery are factored into the solution time, chaotic relaxation has some attractive recovery properties. The tasks that communicate with a failed task can do recovery independently and locally. Furthermore, the information lost by a failed task does not need to be recovered. The calculations can be formulated to proceed and converge to the solution despite failures.

We experimented with super-scalable finite difference algorithms and observed that simple problems, such as 2-D Poisson, converged despite 100 random failures across the machine. However, multiple failures of neighboring tasks, similar to multi-processor node failures, could cause the error of the solution to be significantly higher. However, this can be avoided if the virtual space is not directly mapped to the physical location of processors. Furthermore, connecting tasks randomly can decrease the overall convergence time.

We also experimented with asynchronous multi-grid variants based on the above ideas and this approach also tolerated failures. However, a master that controlled the "V" and "W" cycles was necessary, since the mathematical model of chaotic relaxation between different levels is not yet well understood at this time.

2.2 Global Information

Parallel algorithms where individual tasks require global information include global maximum searches, such as are often used to determine if an iterative algorithm has converged. First, the global maximum needs to be found among the values of all tasks. Then this value needs to be broadcast to all other tasks.

This is a graph problem that can be solved by creating a logical interconnect topology with the property of high probability message delivery despite failures, and that maintains efficient scale invariance to a low degree.

We conducted experiments with different network architectures, such as nearest neighbor, random, mesh and fully connected. We also implemented a broadcast algorithm. Both algorithms, global maximum search and broadcast, worked very well under various failure conditions.

A serious challenge for the global information algorithms, as well as for the finite difference, is algorithm termination. How does each task know when the complete system is stable and all tasks have the correct answer? Only the observing user knows that there are no messages on the network any more and that the system has converged. A global convergence test can solve this problem, but it needs to be either super-scalable or occur very infrequently.

3 Peer-to-Peer Diskless Checkpointing

Problems that cannot be transformed into a super-scalable variant, or where other existing solutions are more efficient, still need to deal with the expected MTTI of 100,000-processor systems.

To address this, we have developed a super-scalable replication technique based on peer-to-peer diskless checkpointing [14], which equips scientific applications with a self-healing capability for fault-tolerance. We assume that on Blue Gene/L like systems local disk storage will no longer be available, due to the associated costs, failure sensitivity and maintenance.

In peer-to-peer diskless checkpointing, every task replicates its own local application state to a set of neighboring tasks using an encoding, such as RAID. The neighbor tasks themselves also replicate their own local application state, each to different sets of neighbor tasks. A scalable peer-to-peer infrastructure of checkpointing tasks is formed with local separation of current application state and multiple redundant backups. The amount of additional information each task needs to hold in its memory is dependent only on the encoding algorithm and on the number of neighbors involved in the replication of the state of one task, i.e. the system-wide degree of fault-tolerance.

The set of neighbor tasks may be derived from the network infrastructure or application algorithm. However, the probability of a failure involving physical neighbors, e.g. multi-processor node failures, may be greater than the probability of a failure involving a set of random or far away neighbors. The physical neighborhood of a task may also change in the case of a restart.

Synchronization of individual checkpoints is not necessary if tasks do not communicate with each other at all or if they do not communicate between

synchronizing checkpoints. The traditional global snapshot method, using a barrier, can be used to synchronously checkpoint all tasks at once. Localized asynchronous checkpointing requires additional message logging to make sure that a consistent application state is being saved. We discussed advantages and disadvantages of both approaches in an earlier paper [14].

In the case of a failure, all surviving tasks roll back to their last checkpoint using a locally maintained copy or the remote backup in the neighboring tasks. All failed tasks are replaced using their last checkpoint from their neighboring tasks. An area of future research would be to identify surviving tasks that do not need to roll back if they are not directly dependent on the failed ones. Furthermore, a localized replay of the message log can eliminate the rollback of surviving tasks all together for a certain set of deterministic scientific applications. While centralized and partially localized rollback strategies and message log replay solutions [15, 16] exist, they currently do not scale to 100,000 processors. Initial work [17] has been done recently to address this issue.

Our experience with peer-to-peer diskless checkpointing shows that it can provide super-scalable self-healing capability for algorithms, such as FFT, where every single task holds important information for calculating the correct result. However, checkpointing and recovery scenarios can generally be very complex, especially when using localized asynchronous mechanisms. Furthermore, an application run still fails if the number of simultaneous failures of neighboring tasks is greater than the system-wide degree of fault-tolerance.

4 100,000-Processor Simulator

While the theoretical analysis of super-scalable algorithms gave us some insight into convergence properties and the probability of achieving the right answer, there is a lot of practical analysis data that can only be acquired by testing the algorithms using a variety of different failure situations.

A 100,000-processor machine was not available at the time of this work, the IBM Blue Gene/L still under development, and software emulation frameworks, such Charm++ [18], did not reached the necessary scale. Therefore, we developed a simulator (Figure 1) that is able to run hundreds of thousands of tasks and supports rapid prototyping. It is designed to test algorithms at very high scale and provide a platform to develop fault-tolerant applications. It is instrumented to mimic different failure modes, but it does not provide performance estimates or analysis of the applications for a particular machine architecture.

The simulator can handle modules written in multiple languages and runs on different operating systems, e.g. Linux and Windows. It is implemented in Java, but also supports C and Fortran using the Java Native Interface. The number of nodes that can be simulated depends on the size of the application being simulated and the power of the hardware the simulator is running on. The simulator is itself a parallel application and can run across a Linux cluster. On a 2 GHz Windows laptop we have simulated 10,000 nodes for a small application. Using a

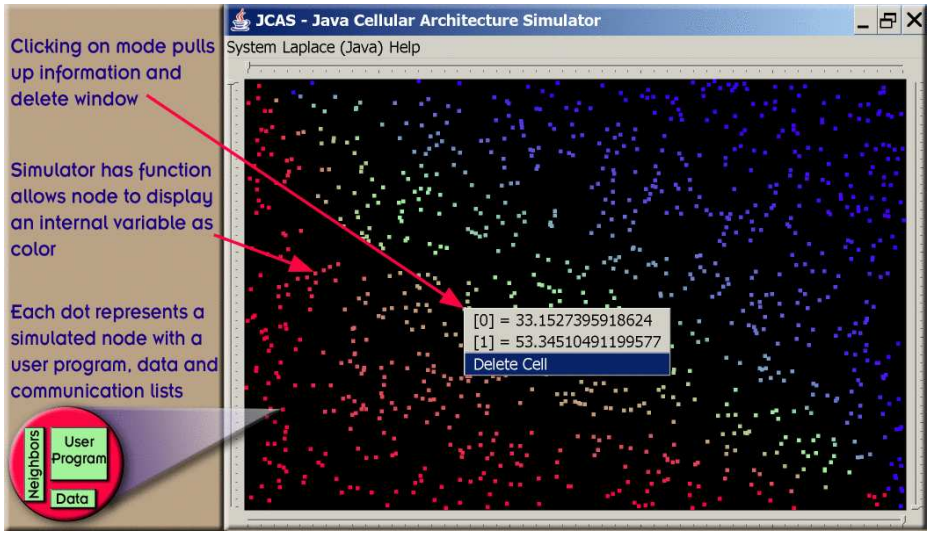

Fig. 1. User Interface to the Simulator

32 processor, large-memory Linux cluster we have simulated half a million nodes running the super-scalable algorithms described earlier in this paper.

The simulated network topology, such as nearest neighbor, mesh, torus and random, can be configured before running an application. The simulator has a number of built in failure modes that the user can specify. It allows the killing of a selected node, block of nodes or a random percentage of nodes in a specified region. The failures are interactively initiated, i.e. the user clicks on a node and kills it, or selects a region and 1% of the nodes in this region die.

5 Conclusions

In this paper we have summarized our recent research at the Oak Ridge National Laboratory in super-scalable algorithms for high-end scientific computing on extreme-scale supercomputer systems with 100,000 processors. We presented the notion of a new class of algorithms called *super-scalable algorithms* that have the properties of *scale invariance* and *natural fault tolerance*. These properties allow scientific algorithms to scale to hundreds of thousands of processors, while maintaining efficiency and fault-tolerance.

We described solutions for two classes of super-scalable algorithms. In the first, the problem can be formulated as some function of a local volume, such as for finite difference applications. In the second, the problem requires global information, like in global maximum searches. We also developed a self-healing FFT based on peer-to-peer diskless checkpointing. Finally, we developed a software simulator that is able to run an enormous number of tasks.

Future research needs to be conducted to further develop appropriate programming models for 100,000-processor machines. Furthermore, scientists will need to rethink the mathematical models used in today's applications to better support the development of super-scalable solutions based on scale invariance and natural fault tolerance.

References

1. Adiga, N.R., et al.: An overview of the Blue Gene/L supercomputer. Proceedings of SC, also IBM research report RC22570 (W0209-033) (2002)
2. Lawrence Livermore National Laboratory, Livermore, CA, USA: ASCII Blue Gene/L Computing Platform at http://www.llnl.gov/asci/platforms/bluegenel
3. Geist, G.A., Engelmann, C.: Development of naturally fault tolerant algorithms for computing on 100,000 processors. (2002) to be published.
4. Bosilca, G., Chen, Z., Dongarra, J., Langou, J.: Recovery patterns for iterative methods in a parallel unstable environment. Submitted to SIAM Journal on Scientific Computing (2005)
5. Space Sciences Laboratory, University of California Berkeley, USA: SETI@HOME at http://setiathome.ssl.berkeley.edu
6. Basney, J., Livny, M.: Deploying a high throughput computing cluster. In Buyya, R., ed.: High Performance Cluster Computing: Architectures and Systems, Volume 1. Prentice Hall PTR (1999)
7. Computer Sciences Department, University of Wisconsin, USA: Condor at http://www.cs.wisc.edu/condor
8. Chazan, D., Miranker, M.: Chaotic relaxation. Linear Algebra and its Applications **2** (1969) 199–222
9. Baudet, G.M.: Asynchronous iterative methods for multiprocessors. Journal of the ACM **25** (1978) 226–244
10. Liu, G.R.: Mesh Free Methods: Moving beyond the Finite Element Method. CRC Press (2002)
11. von Eicken, T., Culler, D.E., Goldstein, S.C., Schauser, K.E.: Active Messages: A mechanism for integrated communication and computation. In: 19th International Symposium on Computer Architecture, Gold Coast, Australia (1992) 256–266
12. Geist, G.A., Beguelin, A., Dongarra, J.J., Jiang, W., Manchek, R., Sunderam, V.S.: PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing. MIT Press, Cambridge, MA, USA (1994)
13. Snir, M., Otto, S., Huss-Lederman, S., Walker, D., Dongarra, J.: MPI: The Complete Reference. MIT Press, Cambridge, MA, USA (1996)
14. Engelmann, C., Geist, G.A.: A diskless checkpointing algorithm for super-scale architectures applied to the fast fourier transform. Proceedings of CLADE (2003) 47–52
15. University of Paris South, France: MPICH-V at http://www.lri.fr/∼gk/MPICH-V
16. Indiana University, Bloomington, IN, USA: LAM-MPI at http://www.lam-mpi.org
17. Chen, Z., Fagg, G.E., Gabriel, E., Langou, J., Angskun, T., Bosilca, G., Dongarra, J.: Building fault survivable MPI programs with FTMPI using diskless checkpointing. Submitted to PPoPP (2005)
18. Zheng, G., Singla, A.K., Unger, J.M., Kale, L.V.: A parallel-object programming model for petaflops machines and blue gene/cyclops. Proceedings of IPDPS (2002)