A Diskless Checkpointing Algorithm for Super-scale Architectures Applied to the Fast Fourier Transform*

Christian Engelmann and Al Geist
Computer Science and Mathematics Division
Oak Ridge National Laboratory
Oak Ridge, TN 37831, USA
{engelmannc,gst}@ornl.gov

Abstract

This paper discusses the issue of fault-tolerance in distributed computer systems with tens or hundreds of thousands of diskless processor units. Such systems, like the IBM BlueGene/L, are predicted to be deployed in the next five to ten years. Since a 100,000-processor system is going to be less reliable, scientific applications need to be able to recover from occurring failures more efficiently. In this paper, we adapt the present technique of diskless checkpointing to such huge distributed systems in order to equip existing scientific algorithms with super-scalable fault-tolerance. First, we discuss the method of diskless checkpointing, then we adapt this technique to super-scale architectures and finally we present results from an implementation of the Fast Fourier Transform that uses the adapted technique to achieve super-scale fault-tolerance.

1 Introduction

In the past four decades, the raw computational power of computer systems increased steadily affirming the initial prediction of Gordon Moore from 1965 that the number of transistors, i.e. the raw computational power, on a chip doubles every eighteen months. With the introduction of parallel computing Moore's Law was even topped by doubling the computational performance of parallel computer systems every 12 months. This trend is not only driven by the achievements in processor technology but also by connecting more and more processors together using low latency/high bandwidth interconnects. The current top system, the Earth Simulator in Japan, has 5120 processors connected with a single stage crossbar (16 GB/s cross section bandwidth) and achieves up to 40 Tera FLOPS.

In the next five to ten years the number of processors in distributed computer systems will rise to tens and even hundreds of thousands in order to keep up with the pace. While the vision of such super-scalable computer systems, like the IBM BlueGene/L [1, 2], attracts more and more scientists for research in areas like climate modeling and nanotechnolgy, existing deficiencies in scalability and fault-tolerance of scientific algorithms need to be addressed soon. Amdahl's Law shows how efficiency drops off as the number of processors increases. A lot of scientific algorithms still do not scale well. Furthermore, the mean time to failure becomes shorter and shorter. A failure may occur every couple of minutes in a system with 100,000 processors. Additionally, computing processors will not have local disk storage any longer due to associated costs, failure sensitivity and maintenance. Network bottlenecks and latencies to stable storage make frequent disk checkpointing (every hour) of applications for fault-tolerance impossible.

In this paper, we describe a super-scalable diskless checkpointing algorithm that can be used to provide scientific algorithms with fault-tolerance for distributed computer systems beyond the 10,000-processor barrier. First we will discuss the existing technique of diskless checkpointing and then its adaptation to super-scale architectures. Finally, we use the Fast Fourier Transform algorithm to demonstrate the super-scalable diskless checkpointing.

2 Diskless checkpointing

Existing techniques of diskless checkpointing [10, 11] are based on a coordinated backup procedure of the global application state, which consists of the state of each process and a log of all in-flight messages. This application state is stored in the memory of dedicated checkpoint processes

*This research was supported in part by an appointment to the ORNL Postmasters Research Participation Program which is sponsored by Oak Ridge National Laboratory and administered jointly by Oak Ridge National Laboratory and by the Oak Ridge Institute for Science and Education under contract numbers DE-AC05-00OR22725 and DE-AC05-00OR22750, respectively.

Proceedings of the International Workshop on Challenges of Large Applications in Distributed Environments (CLADE'03)
0-7695-1984-9/03 $17.00 © 2003 IEEE

using common encoding semantics, such as RAID. By eliminating stable storage, diskless checkpointing reduces overhead and latency allowing more frequent checkpoints and shorter application running time. To avoid the absence of any stable storage backup for long application runs, a two level approach with frequent memory and infrequent disk checkpointing has been used.

In case of single or multiple process failures in a system using diskless checkpointing, all correct processes rollback to their last checkpoint state using either a locally maintained copy or the stored backup from the checkpoint processes. All failed processes are restarted using their last checkpoint states from the checkpoint processes. If no spare processors are available, two or more processes may be assigned to one processor or a dedicated checkpoint processor may become a replacement processor. New checkpoint or spare processors can be added to the system dynamically during runtime when they become available.

Super-scale distributed systems with tens or hundreds of thousands of diskless processor units will likely have short mean times to failure. Checkpointing and restart algorithms will need to recover fast and to progress even while other failures occur. Additionally, complete system backups to remote stable storage may only be performed very infrequently due to I/O limitations. The following adaptation of diskless checkpointing is able to address these problems by embedding a distributed memory checkpointing and restart algorithm into application algorithms.

3 Super-scalable self-healing algorithms

Naturally fault-tolerant algorithms for super-scale architectures [7] are able tolerate failures without requiring notification or recovery from previously saved state. They do this through the mathematical properties of the algorithms themselves. A simple example is an iterative method that requires a little longer to converge if a failure occurs. In this section we describe how algorithms without such properties can be equipped with a scalable replication scheme similar to diskless checkpointing in order to provide a self-healing capability for fault-tolerance.

In a super-scalable self-healing algorithm, every process is only responsible to replicate its own local state to a set of neighbor processes using encoding, such as RAID. The neighbor processes themselves are also replicating their own local state each to different sets of neighbor processes. In contrast to the traditional diskless checkpointing with its global separation of current and backup state, a scalable peer-to-peer infrastructure of checkpointing processes is formed with local separation of current and backup states (see example in figure 1). The amount of additional information each process needs to hold in its memory is only dependent on the encoding algorithm and on the number of

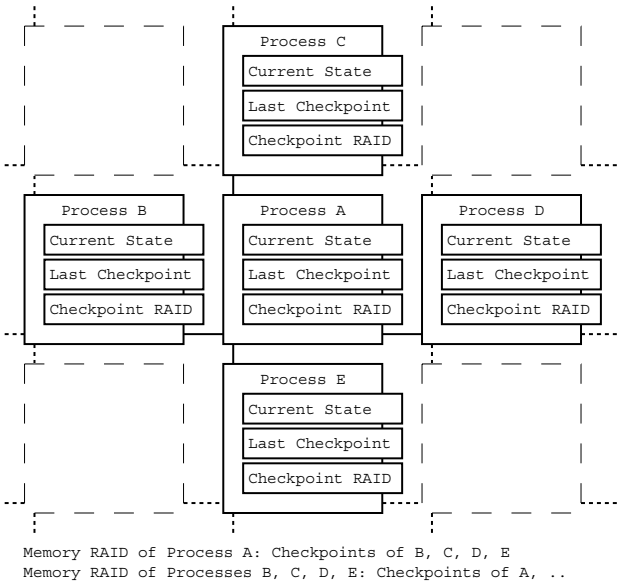

Figure 1. Peer-to-peer infrastructure of checkpointing processes

neighbors involved in the replication of the state of one process, i.e. the system-wide degree of fault-tolerance.

The set of neighbor processes may be derived from the physical network infrastructure, e.g. the set of processors connected to, where it may also be greater than the number of direct connections, e.g. based on a maximum number of hops. The second variant ensures a greater degree of fault-tolerance, while both ensure a minimum of latency and a maximum of bandwidth. However, the probability of a failure involving physical neighbors, e.g. node failures, may be greater than the probability of a failure involving a set of random or far away neighbors.

The set of neighbor processes may also be derived from the logical network infrastructure defined by the application algorithm the checkpointing is embedded in. If the algorithm requires the exchange of process state anyway, why not use it as a replication scheme provided that the states are replicated at multiple processes. This solution has a greater latency and smaller bandwidth if the neighbor processes are not physical neighbors.

Furthermore, the neighborhood of a process may change in case of a failure and restart, since the physical location of the restarted process may be different. In order to provide fault-tolerance, process state needs to be replicated at different physical locations.

Synchronization of checkpointing processes (individual checkpoints) is not necessary if they do not communicate with each other at all or if they do not communicate between synchronizing checkpoints. Once they send each other messages, synchronization is necessary to make sure that pro-

cess restarts are consistent with the algorithm state (coorditated checkpoints). There are two methods for backup coordination: the snapshot method with global synchronization and the record keeping of in-flight messages with local synchronization.

The traditional snapshot method can be used to synchronize the checkpoints of all processes at once. Global barriers are set at a specific algorithm state after a checkpoint was performed requiring each process to wait until all other processes reach the barrier. All process checkpoints are declared valid only after passing the barrier, so that the old checkpoints are valid until all processes have completed their new checkpoints.

When a failure occurs, all correct processes rollback to their last checkpoint state using a locally maintained copy or the remote backup in the neighbor processes. All failed processes are replaced using their last checkpoint states from their neighbor processes. There are no additional group communication algorithms, such as reliable broadcast, needed to ensure the replication procedure, since the global barrier also validates the new checkpoints and therefore acts as an acknowledgement for the replication messages. This method produces p times k replication messages, where p is the total number of processes and k is the degree of fault-tolerance (the average number of neighbors assigned to one process).

The record keeping of in-flight messages is a method to synchronize the checkpoints only of dependent processes. A message is called in-flight once it is sent out and until the receiver returned an acknowledgement. It is stored in the process state for that period. The receiver sends an acknowledgement if it does not need the message anymore to reconstruct its current state from its replicated backup state (usually after a checkpoint).

In case of a failure, all correct processes continue to work. All failed processes are replaced using their last checkpoint states from the neighbor processes. A restarted process may request from another (correct or restarted) process to resend a lost message. Since messages are part of the process state and only deleted once they are not needed anymore, the restart is consistent even if multiple dependent processes fail at once.

In order to ensure the correctness of the replication procedure, the state replication needs group communication algorithms, such as reliable broadcast. This method produces p times k group communication for the replication process, while it doubles the number of application algorithm messages for the record keeping. It should only be used if an application algorithm has very infrequent messages that do not synchronize all processes.

Variations of this algorithm may handle some parts more efficiently. For example, unprocessed received messages may be stored immediately with a checkpoint to release the copies on the sending site. The sending site may automatically resend all unacknowledged messages in the same order after a restart. The restarted process may access its local copy of the replicated state of the sending site to retrieve the message without networking.

Both methods have their advantages and disadvantages. The global system snapshot is easy to implement, but every coordinated checkpoint and every failure results in a system wide synchronization. The record keeping of in-flight messages is asynchronous, but doubles the number of algorithm messages and involves localized group communication algorithms between neighbors.

Existing fault-tolerant message passing software, such as FT-MPI [4] and PVM [6], do support the needed partial system recovery due to their dynamic management of system global message addressing and routing.

In the following sections we are going to show how two different versions of the parallel Fast Fourier Transform can be equipped with the introduced super-scale diskless checkpoint and restart to provide a self-healing capability for fault-tolerance.

4 Parallel FFT

The Fast Fourier Transform (FFT) plays an important role in scientific research areas, such as climate modeling [3, 5]. Scientific applications for large-scale distributed systems widely use parallel variants [8, 9] of the serial algorithm despite the fact that a huge communication overhead, about 50%, may be involved. The FFT algorithm is not naturally fault-tolerant, since all intermediate values are critical to the calculation of the final result. A single process failure affects more than one output value and cannot be recovered without using a previously replicated backup of the process state.

The serial FFT is a very fast algorithm for computing a 1-D Fourier transform in $nlog(n)$ steps. Multi-dimensional FFTs can be performed in a sequence of 1-D transforms by applying them in every dimension. For example, n 1-D transforms of the length m with the stride 1 and m 1-D transforms of the length n with the stride m are carried out for a 2-D transform of an m by n array. The second set of 1-D transforms is basically applied to the transposed result of the first set of 1-D transforms.

There are two common parallel variants of the FFT. The distributed FFT is a parallel version of the 1-D transform. It splits the inner loop over multiple processes and involves crosswise message exchanges between processes for some stages of the FFT. The transposed FFT is a parallel version of a multidimensional transform. It literally transposes the array between local 1-D FFT runs over the rows.

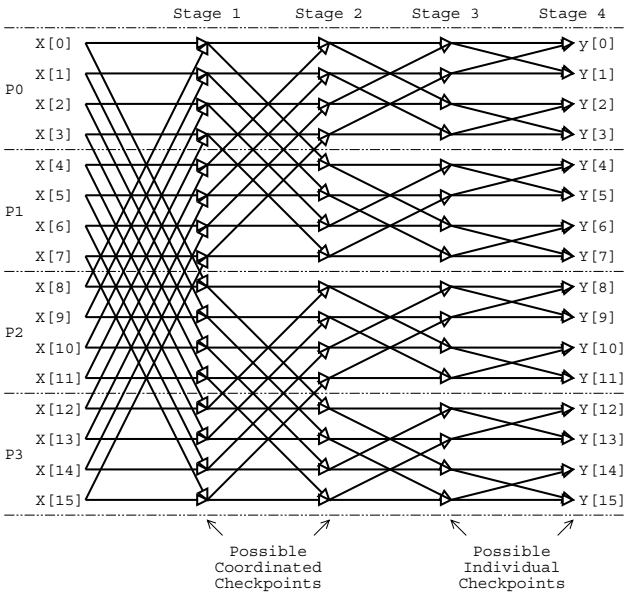

Figure 2. Distributed FFT

5 Fault-tolerant distributed FFT

The serial FFT has $log(n)$ stages of n operators. Before every stage, a crosswise exchange of values is performed between values that are $2log(n)$-stage apart. Assuming a distribution of n values over p processes the communication for n/p values are process-local in the last $log(n/p)$ stages. The distributed FFT consists of two phases (see example in figure 2). The first phase involves the crosswise messaging before every stage resulting in high inter-process dependency. The second phase consists of totally independent computation at all processes and exists only if more than one value is assigned to one process ($n/p > 1$).

In the first phase of the distributed FFT, processes are highly dependent on each other due to the crosswise messaging. A coordinated checkpointing and restart in case of a failure is needed to ensure consistency. The coordination can be performed using either the snapshot method or the record keeping of in-flight messages. However, since heavy messaging by the application algorithm is involved we can show that the record keeping of in-flight messages should not be used in this case.

The traditional snapshot method can be used to regularly checkpoint the global system state at once after a specific number of stages are computed. A global barrier is set after every mth stage is computed and the set of values is replicated. Once every process reaches that barrier the replicated sets of values become valid and the old ones are discarded. In case of a failure, all processes fall back to the last checkpoint using a locally maintained copy or the backup in their neighbors.

Can the record keeping of in-flight messages also be used as an alterative asynchronous coordination method? Yes, since messages are exchanged crosswise before every stage, another exchange of messages after completing a stage can be used to implement the acknowledgement of messages enabling the sending site to delete the message from its memory. All unacknowledged messages and the set of values are replicated at a checkpoint. Processes are free to checkpoint individually.

In case of a failure, correct processes continue to work while failed processes are restarted with their last checkpoint, i.e. with their last set of values. A restarted process always needs to restore lost messages, since the FFT algorithm progresses only after receiving a message. Request messages are sent to processes to retrieve messages until the restarted process is in sync.

Since the record keeping of in-flight messages doubles all application messages to satisfy the acknowledgement scheme and the distributed FFT is a communication bound algorithm, the traditional snapshot method is the way to go in the first phase. Additionally, a race condition exists for restarted processes due to the high volume of messaging and short computation periods. A restarted process may never get in sync and may always request messages. In the worst-case scenario, the original amount of messages is going to be tripled for every faulty process after a restart. Automated sending of lost messages by the sender site can eliminate this race condition.

Since processes do not depend on each other in the second phase of the distributed FFT, individual checkpointing of the current values using a group communication algorithm at the set of neighbor processes can be used. In case of a failure, correct processes continue to work while failed processes are restarted with their last checkpoint.

A simplification of checkpointing and restart may also just implement regular application global snapshots using barriers. A synchronized global checkpoint may still be faster than p times asynchronous replication procedures using group communication.

Variations of the distributed FFT [9] dynamically modify the value assignment to gain multiple blocks of continuous independent parallel computation each followed and preceded by global shuffle communication. Sets of values, each only with internal crosswise exchanges, are assigned to processors due to the value shuffling. Since the global shuffle communication involves synchronization anyway, a system-wide snapshot can be done after each global shuffle and individual checkpoints can be done regularly during the independent computation and right before the next global shuffle. The global shuffle can be combined with the state replication to save communication costs. The barrier for the snapshot after the global shuffle ensures the complete communication step.

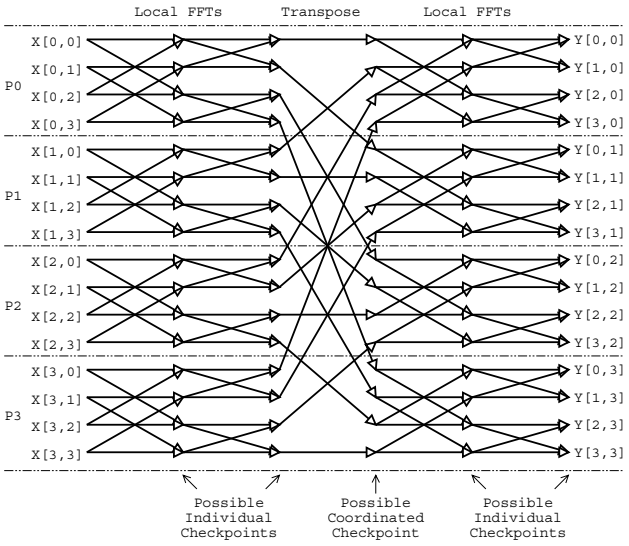

Figure 3. Transposed FFT

6 Fault-tolerant transposed FFT

Multi-dimensional Fourier transforms can be performed as a sequence of 1-D transforms by applying them in every dimension (see example in figure 3). Each processor is assigned to one or more row vectors and runs the serial 1-D FFT. A global matrix transpose is executed to realign the row vectors, so that the next set of 1-D transforms runs over the columns. Sets of serial 1-D FFT transforms are executed in parallel with a matrix transpose for every dimension.

Similar to the shuffled distributed FFT, blocks of independent computation are enclosed by global communication. The computation involves a local serial FFT without any communication. A system-wide snapshot can be taken after each matrix transpose and individual checkpoints can be performed regularly during the independent computation and right before the next matrix transpose.

In case of a failure, all processes fall back to the last checkpoint using a locally maintained copy or the backup in their neighbors. Since an individual checkpoint is taken right before and a snapshot is performed right after the global matrix transpose, any failure during that global communication phase results in a fallback to the state of the finished local FFT. The computation results are already replicated individually and only the global matrix transpose is restarted again. More interestingly, the global matrix transpose can be combined with the state replication to save communication costs.

7 Conclusions

In continuation of our initial research at the Oak Ridge National Laboratory on naturally fault-tolerant algorithms for super-scale distributed systems with 100,000 or more processors, this paper goes a step further and describes how existing scientific algorithms can be equipped with a self-healing capability in order to provide scalable fault-tolerance in such huge computer systems. In the absence of local stable storage and the limited bandwidth to remote stable storage, a peer-to-peer diskless checkpointing algorithm was used to replicate process state at the memory of neighbor processes. The neighbor processes themselves were also replicating their own local state each to different sets of neighbor processes. The fixed number of neighbor processes and their physical location determined the degree of fault-tolerance. Different techniques were illustrated for the checkpointing coordination. Individual checkpointing by processes could only be used if there is no messaging involved. Synchronized checkpointing coordination based on a global application state snapshot proved to be easy to use, while asynchronous checkpointing coordination based record keeping of in-flight messages involved additional messaging overhead. We showed how different versions of the parallel Fast Fourier Transform (distributed and transposed) could be equipped with the introduced super-scale diskless checkpointing algorithm. As a result, scientific applications may be able to run fault-tolerant on super-scale distributed systems. Until the delivery of these systems and the deployment of such software, many open questions remain. We will continue to do research in this area with an emphasis on the further development of application oriented super-scale fault-tolerant algorithms.

References

[1] *The ASCII BlueGene/L Computing Platform*. Lawrence Livermore National Laboratory, Livermore, CA, USA. Available at http://www.llnl.gov/asci/platforms/bluegenel.

[2] N. R. Adiga et al. An overview of the BlueGene/L supercomputer. *Proceedings of SC2002, also IBM research report RC22570 (W0209-033)*, 2002.

[3] J. B. Drake, I. T. Foster, J. G. Michalakes, B. R. Toonen, and P. H. Worley. Design and performance of a scalable parallel community climate model. *Parallel Computing*, 21(10):1571–1591, 1995.

[4] G. E. Fagg, A. Bukovsky, and J. J. Dongarra. Harness and fault tolerant MPI. *Parallel Computing*, 27(11):1479–1495, 2001.

[5] I. T. Foster and P. H. Worley. Parallel algorithms for the spectral transform method. *SIAM Journal on Scientific Computing*, 18(3):806–837, 1997.

[6] G. A. Geist, A. Beguelin, J. J. Dongarra, W. Jiang, R. Manchek, and V. S. Sunderam. *PVM: Parallel Virtual*

Machine: A Users' Guide and Tutorial for Networked Parallel Computing. MIT Press, Cambridge, MA, USA, 1994.

[7] G. A. Geist and C. Engelmann. Development of naturally fault tolerant algorithms for computing on 100,000 processors. *Parallel and Distributed Computing*, 2002. to be published.

[8] A. Gupta and V. Kumar. The scalability of FFT on parallel computers. *IEEE Transactions on Parallel and Distributed Systems*, 4(8):922–932, 1993.

[9] C. F. V. Loan. *Computational Frameworks for the Fast Fourier Transform*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1992.

[10] J. S. Plank, Y. Kim, and J. J. Dongarra. Fault tolerant matrix operations for networks of workstations using diskless checkpointing. *Parallel and Distributed Computing*, 43(2):125–138, 1997.

[11] J. S. Plank, K. Li, and M. A. Puening. Diskless checkpointing. *IEEE Transactions on Parallel and Distributed Systems*, 9(10):972–986, 1998.