Background

Bosilca et al. [1] present an algorithm based fault tolerant scheme for multiprocessor systems that effectively deals with processor loss (aka erasure failures). Their scheme is diskless and relies on checksums. It utilizes (2p-1) checksum processors for every p^2 computing processors.

This method's efficiency grows with the numbers of processors. However, for smaller numbers of cores such as the 8, 16, 32, 64 or 80 cores found on today's multicore processors, at best, 17% of the cores must be allocated to checksum usage. This can be prohibitive for embedded systems with tight power and space requirements.

Our fault tolerant algorithm uses no extra cores for protection against erasure faults but instead uses the algorithm's natural synchronization points to check for core faults and back up important information. Additionally, our system can be used to dynamically add cores to a workload.

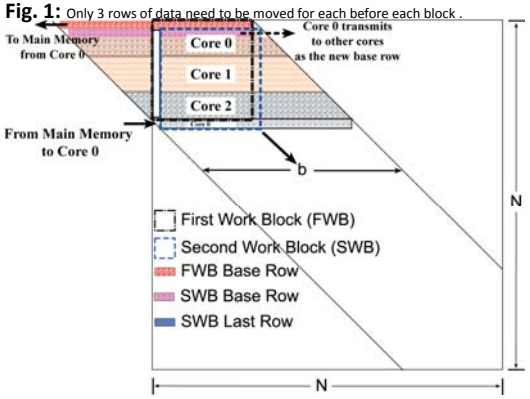
Gaussian Elimination

The in-core algorithm presented in [2, p.195] is implemented for this work. This algorithm operates on banded matrices like the one shown in **Fig. 1**. Everything outside the center band of width **b** is zero so no computation is performed on data in this area. Everything within the band is computed on. The algorithm initially distributes the rows among the cores in a block-round-robin manner so that each core receives a series of contiguous rows as seen in **Fig. 1**.

Elimination starts from the upper left hand corner of the matrix and proceeds to the lower right hand corner of the matrix. Each elimination step starts with a base row which is used to eliminate the left most column in the b/2 by b/2 sized block directly below the base row.

When moving from the first work block to the second work block, the first work block's base row is stored in main memory. The new base row is broadcast to all the cores, and a new row of data is brought onto the chip. This process is seen in **Fig. 1**. This algorithm yields a compute to memory access ratio of: **Compute/memory access = $(b/2)^2/(3b) = b/12$**

Fig. 1: Only 3 rows of data need to be moved for each before each block .



Fault Tolerance

Fault tolerance (FT) is added to the algorithm by having the cores back up their rows to memory periodically. This FT algorithm implementation, while based on the FT algorithm for Gaussian elimination for the Cell Broadband Engine presented in [2], varies significantly in implementation and has more capabilities. The FT algorithm in [2] could only redistribute rows upon an erasure fault. However, this implementation, in addition to being able to redistribute the workload in response to a failure, can add cores independent of a failure and as part of a response to a failure.

The fault tolerance capability takes advantage of the natural serialization points of the algorithm to check for core failures. While checking for failures is free, fault tolerance requires that the data be backed up in main memory. If these accesses can be hidden with compute, then the fault tolerance incurs no performance costs. No attempt was made to hide these access for the present implementation. This optimization has been left for future work.

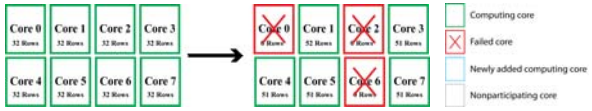
Algorithm 1: Rough pseudo-code for the algorithm

```
while(present_base_row < last_row){
  eliminate_on_my_rows(present_base_row);
  back_up_data_to_memory(); // introduced for fault tolerance
  synchronize_with_other_cores() // share the next base_row information here
  present_base_row++;
}
```

Single or multiple core failures:

In this situation N cores are computing and M < N cores fail. The rows associated with the failed cores are redistributed among the remaining cores. If necessary, the remaining cores perform any lost computation needed on the defunct cores' rows, before continuing with the solve.

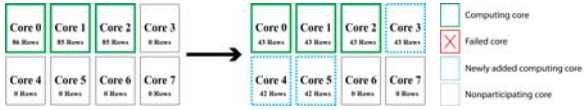
Fig. 2: Eight cores are computing and then 3 cores fail. The failed cores' rows are redistributed among the remaining cores.



Single or multiple core additions:

In this situation cores are added to a workload. The original cores give up enough rows to the new cores so that the work done by each core is about equal.

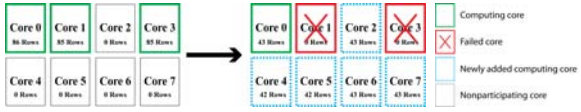
Fig. 3: Three cores are computing and then three more cores are added mid-compute.



Simultaneous core failures and additions:

When a failure occurs, the failed cores are first removed, then the new cores are started, and finally the workload is redistributed.

Fig. 4: Three cores are computing, two of the cores fail, and the system responds by adding four new cores.



Performance

Forward elimination performance without fault tolerance achieves a peak of around 4 Gflops on an IBM QS22. With fault tolerance activated that performance drops to around 0.7 Gflops. Around 0.14 Gflops was achieved using 'lu' in Matlab on a Dell M6400 with a 2.67 Ghz Core 2 Duo T9550 with 'b' = 768. The drop in performance between the non-FT and FT implementations is due to the large number of new memory accesses introduced by the FT algorithm. This drop in performance can be mitigated in future implementations through the use of Cell coding techniques.

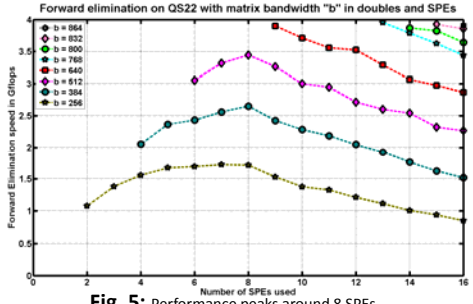


Fig. 5: Performance peaks around 8 SPEs.

Further Applications

Preemptive/Cooperative multitasking:

As proposed in [2], FT can be used as the basis for a preemptive/cooperative multitasking environment for multicore systems. This is because each application knows what to do when its number of available cores changes. Instead of having to remove a task from all the cores on which it is running, a few cores can be liberated and the liberated cores assigned to a new task. If the new task exits before the original task has completed, the cores used by the new task can be re-assigned to the original task.

Thermal Management:

In [3], it was found that there can be significant variation in thermal performance for cores on the same chip. Furthermore, there can be significant thermal variation for the same part on different cores when running the same tasks. For example, during their experiments with a two core test chip, when running the same tasks, a 7.5°C difference between the load store units of the two cores was measured.

The presented fault tolerant scheme can be used to improve energy efficiency when combined with core thermal information. By monitoring the thermal performance of the cores during computation, it can be determined if any of the computing cores has a temperature significantly greater than any of its partner cores. Using the "Simultaneous core failures and additions" capability of the algorithm, the hot core can be shut down and its workload moved to a new different cooler core or redistributed among the remaining cores, thereby improving energy efficiency.

References

[1] G. Bosilca and R. Delmas and J. Dongarra and J. Langou, "Algorithmic Based Fault Tolerance Applied to High Performance Computing," *LAPACK Working Note #205, June 2008.*

[2] J. Geraci, *A comparison of parallel Gaussian elimination solvers for the computation of electrochemical battery models on the cell processor, Thesis Ph.D., MIT, 2008.*

[3] E. Kursun and C. Chen-Yong, "Thermal variation characterization and thermal management of multicore architectures," *Micro, IEEE, Vol 29, No. 1, Jan 2009.*

SERC SQUARE ENIX RESEARCH CENTER RESEARCH TO WIN.