Towards a Distributed GPU-Accelerated Matrix Inversion

Gerardo Ares[1,2] and Pablo Ezzatti[2] and Enrique S. Quintana-Orti[3]

[1]Bull, São Paulo-Brazil gerardo.ares@lam-bull.com
[2]Instituto de Computación, Universidad de la República, Montevideo-Uruguay gares,pezzatti@fing.edu.uy
[3]Dpto. de Ingeniería y Ciencia de Computadores, Universidad Jaume I, Castellón-Spain quintana@icc.uji.es

Mendoza-Argentina - 29th-30th July 2013

Outline

Motivation

- Several examples of the need of matrix inversion are earth-sciences, matrix sign function for spectral decomposition, low rank radiosity technique for computer graphics and many others
- The computation of dense matrix inversion requires an important computational effort in terms of execution time and memory
- Distributed-memory platforms and GPGPU are used to tackle this problem

Outline

Matrix Inversion: Methods and Parallelization

- Conventional strategy is based on Gaussian elimination (LU factorization)
  LAPACK provides an implementation with `getrf` and `getri` functions for shared-memory platforms, while ScaLAPACK is an extension of LAPACK for distributed memory
- Based on Gauss-Jordan elimination algorithm (GJE)
- The GJE algorithm casts the bulk of computation in terms of matrix-matrix products
- The number of floating-point arithmetic operations is $2n^3$

Matrix Inversion: Methods and Parallelization

```
Algorithm: $[A] := \mathrm{GJE\_BLK}(A)$

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$
    where $A_{TL}$ is $0 \times 0$
while $m(A_{TL}) < m(A)$ do
    Determine block size $b$
    Repartition
        $\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$
        where $A_{11}$ is $b \times b$

    $\begin{bmatrix} A_{01} \\ A_{11} \\ A_{21} \end{bmatrix} := \mathrm{GJE\_UNB}\left(\begin{bmatrix} A_{01} \\ A_{11} \\ A_{21} \end{bmatrix}\right)$     Unblocked Gauss-Jordan
        $A_{00} := A_{00} + A_{01}A_{10}$                   Matrix-matrix product
        $A_{20} := A_{20} + A_{21}A_{10}$                   Matrix-matrix product
        $A_{10} := A_{11}A_{10}$                            Matrix-matrix product
        $A_{02} := A_{02} + A_{01}A_{12}$                   Matrix-matrix product
        $A_{22} := A_{22} + A_{21}A_{12}$                   Matrix-matrix product
        $A_{12} := A_{11}A_{12}$                            Matrix-matrix product

    Continue with
        $\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$
endwhile
```

Matrix Inversion: Methods and Parallelization

- The content of A is overwritten with the values of the inverse
- Partial pivoting to ensure ('in practice') numerical stability
- At each iteration an unblocked Gauss-Jordan is calculated and then six matrix-matrix products are made

Outline

Distributed Implementation of Matrix Inversion

- As GJE performs the same number of operation on each iteration, a parallel algorithm does not require a cyclic distribution of the data
- Distributed the matrix in c nodes of the cluster following a column block data layout, with $d = n/c$ consecutives columns assigned to each node
- At each iteration the $\mathrm{GJE\_{\scriptsize UNB}}$ is computed at the node where the current column panel resides
- After that, the factored column panel and the local pivot indexes are broadcasted to the remaining nodes to compute three matrix-matrix products
- The node with the current column panel will compute six matrix-matrix products

Distributed Implementation of Matrix Inversion

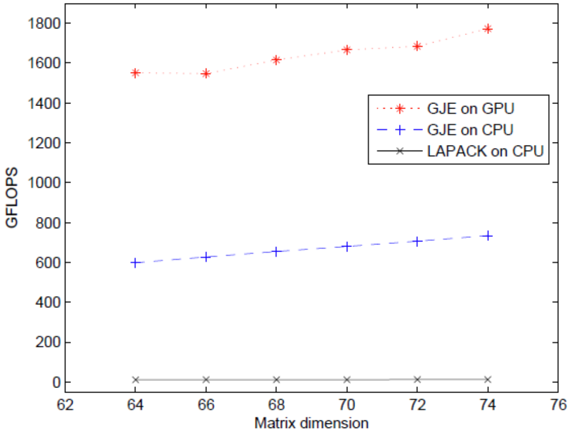
- Two variants of the GJE were implemented:
    - Matrix inversion on homogeneous CPU based clusters
      All computation is done on CPU
    - Matrix inversion on hybrid CPU+GPU based clusters
      The GJE_UNB computation is done in GPU while the matrix-matrix products are done in GPGPU
- The MPI library was used for distributed programming

Outline

Experimental results

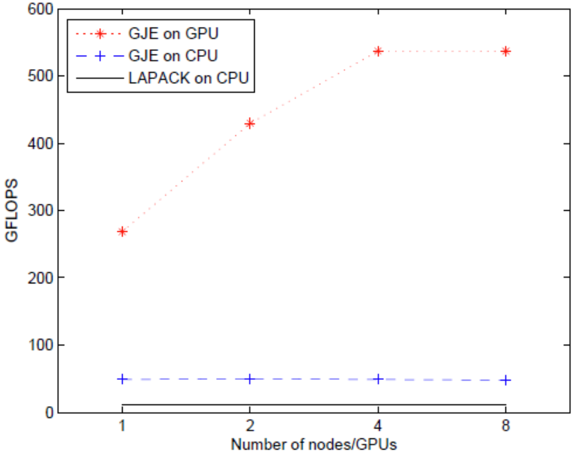
- All the experiments were performed using IEEEsingle precision on two different platforms:
  - CPU cluster: 8 computes nodes with 2x Intel Sandy-Bridge processors at 2.70GHz (8 cores per socket) and 64 GB of RAM memory, Infiniband QDR network
    bullx Linux Server release 6.3 (v1), Intel v12.1.4 compiler, Intel MKL library (for BLAS) and bullxmpi v1.2.4.1 (MPI v2 implementation)
  - GPU+CPU cluster: 4 computes nodes with 2x Intel Nehalem E5640 processors at 2.67GHz (4 cores per socket) and 24 GB of RAM memory, Infiniband QDR network
    Each node with 2x NVIDIA Tesla M2050(Fermi) GPUs
    bullx Linux Server release 6.3 (v1), Intel v12.1.4 compiler, Intel MKL library (for BLAS) and bullxmpi v1.2.4.1 (MPI v2 implementation), CUDA v4.0.17

Experimental results



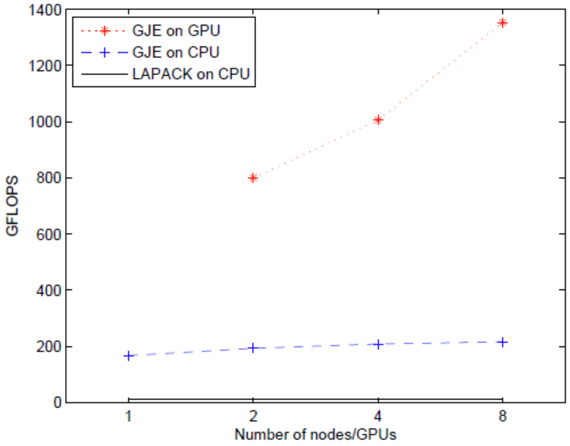
- GJE on GPU speed-up to 2.6x with respect to CPU variant

Experimental results



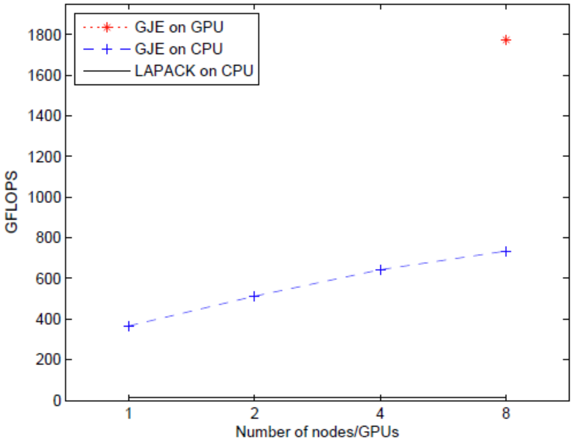
- Scalability for matrix size 10K

Experimental results



- Scalability for matrix size 32K

Experimental results



- Scalability for matrix size 74K

Outline

Concluding remarks

- Initial implementation of a distributed GPU based for matrix inversion
- Significant improvements on the performance, with speed-ups of up to 2.6x compared to CPU-based versions
- GPU-based proposal exhibits a reasonable degree of weak scalability and acceptable values of strong scalability
- Use of several GPUs increases the aggregated memory space

Future work

- Use of larger platforms to reduce computational time and tackle bigger problems
- Migrate the code to a 2D block data distribution
- Incorporate GPU Direct to advance in the scalability of the proposal

Thank you

Questions?