PGI insider

Technical News from The Portland Group

November 2010

Using MAGMA With PGI Fortran

by S. Tomov, M. Faverge, P. Luszczek, and J. Dongarra
The Innovative Computing Laboratory
University of Tennessee, Knoxville

Overview

The goal of the Matrix Algebra on GPU and Multicore Architectures (MAGMA) project is to create a new generation of linear algebra libraries that achieves the fastest possible time to an accurate solution on heterogeneous/hybrid systems, using all available processing power within given energy constraints. The main focus is the development of a dense linear algebra library for multicore+GPU systems. MAGMA is designed to be similar to LAPACK in functionality, data storage, and interface thus allowing scientists to effortlessly port any of their LAPACK-relying software components to take advantage of the new hybrid architectures.

MAGMA is being designed to run on homogeneous x86-based multicores and to take advantage of GPU components if available. This is achieved by developing a class of hybrid algorithms that split the computation into tasks of varying granularity (e.g., large for available GPUs) and scheduling their execution over the available hardware components.

MAGMA 1.0

MAGMA 1.0 concentrates on fundamental linear algebra algorithms for multicore systems employing a single GPU. This includes both the basic one and two-sided matrix factorizations, as well as linear systems and eigen/singular-value solvers based on them. Routines are provided in four levels of precision—single, double, single complex, and double complex. Linear systems solvers are provided in both working precision and mixed-precision using iterative refinement. In addition, MAGMA 1.0 includes MAGMA BLAS, a subset of optimized CUDA BLAS for NVIDIA GPUs, including support for the newest Fermi GPUs. The MAGMA matrix-matrix multiplication routine (GEMM) for Fermi is now used in CUBLAS 3.2. Other routines in MAGMA BLAS that are not included in CUBLAS outperform CUBLAS significantly. For example GEMMs with particular matrix dimensions, symmetrix matrix-vector products, triangular matrix solvers, etc. MAGMA 1.0 is an open source project. The current release is available through the MAGMA homepage.

Most MAGMA routines have at least two interfaces—CPU and GPU—depending on where the input data and the result is expected. For example, in the GPU interface the input data is copied into GPU memory and the result is generated and stored on the GPU as well. The CPU interface is similar but the input data and the results are kept in the CPU memory. The CPU interface provides the easiest way to accelerate software relying on LAPACK using GPUs as no GPU knowledge is needed to accomplish it. MAGMA LAPACK routines are called by using the `magma_` prefix.

MAGMA and Fortran

MAGMA is written in C but it has Fortran bindings to simplify its use from Fortran programs. MAGMA uses the CUDA bindings available in the file fortran.cpp from NVIDIA's CUDA SDK. This file is included with the MAGMA distribution. Below is a simple example program using LU factorization in double precision. Also known as `dgetrf` in LAPACK.

```

!-------- include PGI MAGMA module

            use magma

!-------- declarations

            double precision,   allocatable, dimension(:) :: h_A, h_R

            integer, allocatable, dimension(:) :: ipiv

            integer :: d_A

            . . .

!--------- initialize CUBLAS
```

```
              call cublas_init()
```

```

```

```
!---------- allocate CPU memory
```

```
              allocate(h_A(M*N), h_R(M*N), ipiv(min(M,N)))
```

```

```

```
!---------- allocate GPU memory
```

```
               call cublas_alloc(M*N, sizeof_double, d_A)
```

```

```

```
!---------- Initializa the matrix
```

```
               do i=1,n*n
```

```
                  call random_number(rnumber)
```

```
                  h_A(i) = rnumber(1)
```

```
                  h_R(i) = rnumber(1)
```

```
                end do
```

```

```

```
!-----------  d_A = h_A
```

```
                call cublas_set_matrix (M, N, sizeof_double, h_A, lda, d_A,
lda)
```

```

```

```
!----------- call MAGMA GPU interface
```

```
                call magma_dgetrf_gpu(M, N, d_A, lda, ipiv, info)
```

```

```

```
!----------- call LAPACK
```

```
                call dgetrf(M, N, h_A, lda, ipiv, info)
```

```
                . . .
```

```
!------------ Free GPU memory and exit

                call cublas_free(d_A)

                call cublas_shutdown()

                end
```

We then compiled this routine using PGI Fortran:

```
% pgfortran -O3 -DADD_  -I../include -I/usr/local/cuda-3.2/include -c \

  testing_sgetrf_gpu_f.f  -o  testing_sgetrf_gpu_f.o

% pgfortran -O3 -DADD_ -fPIC testing_sgetrf_gpu_f.o fortran.o -o \

  testing_sgetrf_gpu_f -L../lib -lcuda -lmagma -lmagmablas \
```

```
-L/home/tomov/intel/mkl/10.0.1.014/lib/em64t -L/usr/local/cuda-3.0/lib64 \
```

```
-lmkl_em64t -lguide -lpthread -lcublas -lcudart -lm
```

Performance

Figure 1 shows the performance of the MAGMA LU in double precision on an NVIDIA Fermi GPU (C2050). The result is compared with the performance of state-of-the-art packages running on multicore systems. In this case, we use a 48 core AMD-based multicore system (Istanbul processors). The two systems have the the same theoretical peak throughput, in this case around 500 GFlop/s in double precision. The result shows that LU on Fermi achieves better efficiency. Moreover, it is worth mentioning that the cost of the multicore system is an order of magnitude higher than the GPU system ($30,000 vs $3,000).

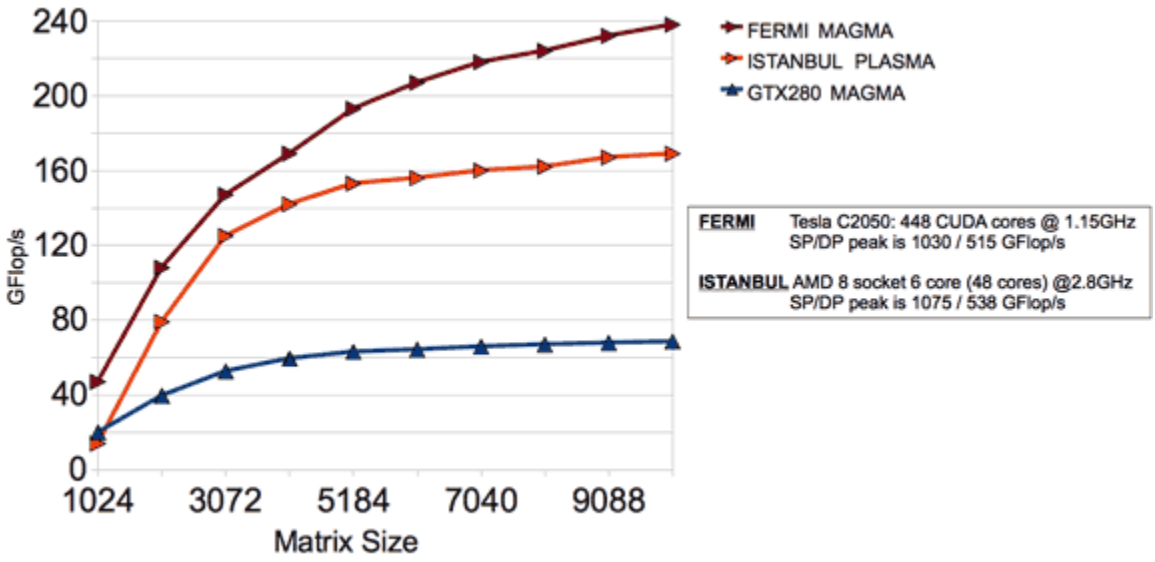

Figure 1. Performance of the MAGMA LU factorization in double precision on a Fermi (C2050) GPU.

Current and Future Work

Current MAGMA work is on integrating it with tools and libraries like PLASMA to more efficiently use multicore hosts, StarPU to support heterogeneous nodes featuring multi-GPUs and multicore, and DAGuE for hybrid manycore+GPUs distributed systems. The MAGMA 1.0 release will be made generally available at SC 2010.