Solving PDEs with PGI CUDA Fortran
Part 3: Linear algebra. Laplace's equation

Outline

Compute- and memory-bound kernels. Matrix multiplication. Optimized libraries for linear algebra. Direct and iterative methods for linear algebraic equations. Laplace's and Poisson's equations in 1D. Direct solution and Jacobi and Gauss-Seidel iterations.

Compute-bound and memory(-bandwidth)-bound kernels

Definitions

number of floating-point operations for SP (single) or DP (double precision)	F [flop]
floating-point operations per second	dF [flop/s]
number of transferred bytes or words	B [byte] or W [word] W=B/4 for SP, W=B/8 for DP
memory bandwidth per second	dB [bytes/s] or dW [words/s]
float:byte ratio	F/B or dF/dB [flop/bytes]
float:word ratio	F/W or dF/dW [flop/word]
relation	F/W = 4 F/B

Compute-bound and memory(-bandwidth)-bound kernels

Theoretical hardware limits	dF SP [Gflop/s]	dB [GB/s]	dF/dB [flop/byte]
Tesla C2070	1030	144	7.2
GeForce GTX 470	1089	134	8.1
GeForce GTX 260	715	112	6.4
GeForce GT 425M	215	25.6	8.4

Theoretical limits in basic linear-algebra algorithms

	F	W	F/W	F/B
dot-product of vectors of n elements	2n	2n+n = 3n	2/3	2.7
matrix-vector product	n . 2n	$n^2+n+n^2 \sim 2n^2$	1	4
matrix-matrix product	n^2 . 2n	$2n^2 + n^2 = 3n^2$	2n/3	2.7 n

i.e., only the matrix multiplication provides the flop:byte ratio large enough for GPUs and therefore can be compute-bound, while BLAS 1+2 algorithms are memory-bound

Matrix multiplication

see PGI CUDA Fortran User Guide, chapter Examples for the source code
see Volkov's papers for a story of developing matmul on GPUs

Optimized libraries for linear algebra

BLAS library (Basic Linear Algebra Subroutines)
for summs, scaling, dot products, matrix multiplication etc.

Levels 1 (vector-vector), 2 (matrix-vector), 3 (matrix-matrix)

in single precision (prefix s), double precision (d), complex SP (c), complex DP (z)

e.g., L1: saxpy alpha*x(1:n)+y(1:n)
L3: dgemm alpha*A(1:m,1:k)*B(1:k,1:n)+beta*C(1:,1:n)

a port for GPUs by NVIDIA: the CUBLAS library

Optimized GPU libraries for linear algebra

LAPACK library (Linear Algebra PACKage)
for solving linear algebraic equations by direct methods, for eigenvalue problems

includes
– BLAS Levels 1, 2, 3
– direct solvers of linear algebraic systems with general, band-diagonal matrices, symmetric positively definite matrices etc.
– algorithms: LU, QR and Cholesky factorization, singular value decomposition (SVD) etc.
– dense, banded and other matrices

a port always available in optimized general-purpose libraries:
MKL, IMSL, NAG
variants for sparse matrices and for parallel computing

for GPUs: packages CULA tools (for fee), MAGMA (for free)

Optimized GPU libraries for linear algebra

Running pgfortran with CUBLAS

CUBLAS – a part of CUDA Toolkit, i.e. recent versions: 3.1, 3.2, 4.0
packed in the directory tree of PGI
works with dmem arrays, can be linked with gfortran/g95/ifort
    (examples in App. B of CUBLAS User Guide),
    but with PGI it's simple (examples in CUDA Fortran SDK in PGI tree and now)
example with GEMM: TestCUBLAS.f90
    goals: interoperability of Fortran and C, Fortran interface to overloaded
        subroutines, random numbers

Running pgfortran with CULA tools

CULA tools – Basic version free for SP, Premium version with DP and more routines
    – recent versions R11 (for CUDA 3.2), R12 (for CUDA 4.0)
    – Fortran-relevant interfaces:
        Fortran for host-mem arrays, Device for dmem arrays
performance graphs
examples with GEMM: TestCULA.f90
    goals: interface to CULA functions, CULA initialization, inquiries and shutdown

Direct and iterative methods for linear algebraic equations

Linear algebraic systems

$$Ax = b, \qquad a_{ij}x_j = b_i, \quad i,j = 1,\ldots,n$$

Direct methods

solve for a unique solution (if it exists) in ~ n^3 operations (for dense matrices)
i.e., for $n=10^3$: number of operations ~ 10^9, for $n=10^4$: ~ 10^{12} operations, etc.
no additional information necessary

LU factorization for general (dense) matrices

a) decomposition A = L . U ,
where L is a lower triangular matrix, U an upper triangular matrix
b) solving to algebraic equations L . y = b for vector y
c) solving to algebraic equations U . x = y for vector x
variants for band diagonal (banded) matrices available

Cholesky decomposition for symmetric positive definite matrices

possible to decompose into the form $A = L \cdot L^T$

Direct and iterative methods for linear algebraic equations

Iterative methods

solve for approximate solutions iteratively

$$x^{n+1} = Hx^n + g$$

i.e., an initial approximation x^0 must be provided
and matrix-vector multiplication is performed in each iteration

– the only way when the matrix A is large
– typical requirements for success:
the iteration matrix is sparse
the iterations converge rather fast
e.g., for ~ n operations for matrix-vector multiplications
and ~ n or ~ const number of iterations,
the total number of operations may be linear (~ n)
or quadratic (~ n^2) function of n
– methods: Jacobi, Gauss-Seidel, successive overrelaxation (SOR),
conjugate gradient method (CGM), multigrid method (MG) etc.

Laplace's and Poisson's equations in 1D

Equation and boundary conditions

the second-order differential equation for a real function u(x) of one real variable x

$$\Delta u(x) \equiv \frac{d^2u(x)}{dx^2} \equiv u''(x) = f(x)$$

the right-hand side

Laplace: $f(x) = 0$

Poisson: $f(x) \neq 0$

boundary conditions

Dirichlet: $u(x_0) = u_0, \qquad u(x_{\max}) = u_{\max}$

Neumann: $u'(x_0) = u'_0 \quad \text{or } u'(x_{\max}) = u'_{\max}$

(not simultaneously at both ends)

i.e., the boundary value problem (BVP) for the elliptic differential equation

Laplace's and Poisson's equations in 1D

Features of the solutions to the Laplace's equation (i.e., harmonic functions)

the maximum principle: extremes of u(x) always at the boundary

the mean value theorem: integral over a ball (1D: interval) is proportional to the value in the center of the ball

in 1D: $$u(x_S) = \left(\int_{x_0}^{x\max} u(x)\, dx \right) / \left(x_{\max} - x_0 \right)$$

Laplace's and Poisson's equations in 1D

Analytical solutions

Laplace's equation:

$$u(x) = bx + c$$

Poisson's equation with constant f(x) = a:

$$u(x) = ax^2/2 + bx + c$$

Poisson's equation with arbitrary f(x):

$$u(x) = \iint f(x) + bx + c$$

where b and c can be obtained from the two boundary conditions

Laplace's and Poisson's equations in 1D

Discretization

the equidistant grid

$$
\begin{aligned}
&x_j, \quad j = 0, \ldots, J+1 \\
&x_j = x_0 + j\,dx, \quad dx = (x_{J+1} - x_0)/(J+1) \\
&u_j \equiv u(x_j)
\end{aligned}
$$

the centered 2nd-order finite-difference scheme for the 2nd derivative (FD2)

$$
u''(x_j) \approx \frac{u_{j-1} - 2u_j + u_{j+1}}{dx^2}
$$

the left- and right-hand 1st-order finite-difference schemes for the 1st derivative,
needed for the Neumann boundary conditions

$$
u'(x_0) \approx \frac{u_1 - u_0}{dx}, \quad u'(x_{J+1}) \approx \frac{u_{J+1} - u_J}{dx}
$$

Laplace's and Poisson's equations in 1D

Discretized system of linear algebraic equations

– with the Dirichlet boundary conditions at both ends

$$\begin{aligned} 2u_1 - u_2 &= -f_1\, dx^2 + u_0 \\ -u_{j-1} + 2u_j - u_{j+1} &= -f_j\, dx^2, \qquad j = 1+1, \ldots, J-1 \\ -u_{J-1} + 2u_J &= -f_J\, dx^2 + u_{J+1} \end{aligned}$$

i.e., the matrix of the system takes the tridiagonal (symmetric positive definite) form

$$\begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & . & . & . & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ . \\ u_{J-1} \\ u_j \end{pmatrix} = \begin{pmatrix} -f_1\, dx^2 + u_0 \\ -f_2\, dx^2 \\ . \\ -f_{J-1}\, dx^2 \\ -f_J\, dx^2 + u_{J+1} \end{pmatrix}$$

– with the Neumann boundary condition at one or the other end,
the first or the last equation is different

$$\begin{aligned} u_1 - u_2 &= -f_1\, dx^2 - u'_0\, dx \\ -u_{J-1} + u_J &= -f_J\, dx^2 + u'_{J+1}\, dx \end{aligned}$$

or

Laplace's and Poisson's equations in 1D

Direct solution

Linear algebraic equations with tridiagonal matrix

(see Numerical Recipes chapter 2.3)

a) a loop to eliminate of subdiagonal elements

b) a loop to eliminate superdiagonal elements ("backsubstitution")

Example of direct solution to 1D Laplace's and Poisson's equations

based on the serial routine tridag (Numerical Recipes Chapter 2.4)

Laplace's and Poisson's equations in 1D

Iterative solution

Decomposition A = L + D + U with L lower triangular,
D diagonal and U upper triangular matrix

Then, (L + D + U) . x = b

can be rewritten into the form suitable for iterations,

$$x = D^{-1} (b - (L + U) . x)$$

or, for each row,

$$x_i = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} A_{ij} x_j - \sum_{j=i+1}^{J} A_{ij} x_j \right), \quad i = 1, \ldots, J$$

Laplace's and Poisson's equations in 1D

Jacobi iterations

the iteration index n is appended to both L . x and U . x terms

$$x^{n+1} = D^{-1}\left(b - (L+U)x^n\right)$$

$$x_i^{n+1} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} A_{ij}x_j^n - \sum_{j=i+1}^{J} A_{ij}x_j^n\right), \quad i = 1,\ldots,J$$

Gauss-Seidel iterations

the iteration index n is appended to U . x term only, the n+1 goes to L . x

$$x^{n+1} = (L+D)^{-1}\left(b - Ux^n\right)$$

$$x_i^{n+1} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} A_{ij}x_j^{n+1} - \sum_{j=i+1}^{J} A_{ij}x_j^n\right), \quad i = 1,\ldots,J$$

Laplace's and Poisson's equations in 1D

Features:

– updates in Jacobi iterations have to be stored into new memory positions
and can therefore be performed in parallel

– Gauss-Seidel iterations update the existing memory positions and are supposed
to be performed serially

– Gauss-Seidel is proved to be slightly more accurate for some matrices

Examples of Jacobi and Gauss-Seidel-like iterations for 1D Laplace's equation

Links and references

Libraries

CUBLAS Library User Guide
http://developer.nvidia.com/nvidia-gpu-computing-documentation
CULA Programmers Guide and Reference Manual
http://www.culatools.com/features/performance
MAGMA, Matrix Algebra on GPU and Multicore Architectures
http://icl.cs.utk.edu/magma/
Calling CUBLAS from CUDA Fortran, 2010
http://cudamusing.blogspot.com/
Humphrey J., Spagnoli K., Using the CULA GPU-enabled LAPACK Library with CUDA Fortran, PGInsider, 2010
http://www.pgroup.com/lit/articles/insider/v2n3a5.htm
Tomov S. et al., Using MAGMA with PGI Fortan, PGInsider, 2010
http://www.pgroup.com/lit/articles/insider/v2n4a4.htm
Toepfer C., Using GPU-enabled Math Libraries with PGI Fortran, PGInsider, 2011
http://www.pgroup.com/lit/articles/insider/v3n1a5.htm

Links and references

Matrix multiplication

Volkov V., Demmel J. W., Benchmarking GPUs to Tune Dense Linear Algebra, 2008
http://www.cs.berkeley.edu/~volkov/

Volkov V., Demmel J. W., LU, QR and Cholesky factorizations using vector capabilities of GPUs, 2008

Nath R. et al., An Improved MAGMA GEMM for Fermi GPUs, 2010
http://icl.cs.utk.edu/projectsfiles/magma/pubs/fermi_gemm.pdf

Numerical methods

Press W. H. et al., Numerical Recipes in Fortran 77: The Art of Scientific Computing, Second Edition, Cambridge, 1992
Chapter 2.3: LU decomposition and its applications
Chapter 2.4: Tridiagonal and band diagonal systems of equations
Chapter 19.0: Partial differential equations – Introduction
http://www.nr.com, PDF available at http://www.nrbook.com/a/bookfpdf.php