High Performance Linpack – HPLinpack

- HPLinpack implements the Gaussian elimination with partial pivoting (paper and pencil benchmark)
- Performance mostly impacted by computation
  - $\frac{2}{3}n^3 + 2n^2$ floating point double precision operations and $O(n^2)$ communication time
- Represents a small class of applications
  - Blocked multiply-add with high cache locality
  - Upper bound on achievable performance
- Reference implementation available
  - Two dim. block-cyclic data distribution
  - LU factorization with partial row pivoting with multiple look-ahead depths
  - Pivot search and column broadcast combined
  - Backward substitution with look-ahead of depth 1
  - Written in C with MPI and BLAS library
HPLinpack

HPL solves a linear system of order \( n \):

\[
A \times x = b
\]

- Compute LU factorization with partial pivoting of \( n \)-by-\((n+1)\) matrix:
  
  \[
  [A,b] = [[L,U],y]
  \]

- Lower triangular factor \( L \) is applied to \( b \) as factorization process, solution \( x \) is obtained by solving the upper triangular system:

  \[
  U \times x = y
  \]

Array \( b \) is part of matrix \( A \) => HPL is not a general LU factorization

Gaussian Elimination (N=3 Example)

\[
\begin{align*}
2x + 4y - 2z &= 2 \\
4x + 9y - 3z &= 8 \\
-2x - 3y + 7z &= 10
\end{align*}
\]

- Using the first equation to eliminate \( x \) from the next two equations
Gaussian Elimination (cont.)

\[ 2x + 4y - 2z = 2 \]
\[ y + z = 4 \]
\[ y + 5z = 12 \]

- Using the second equation to eliminate \( y \) from the third equation

Solving Triangular Systems

- We now have a triangular system which is easily solved using a technique called **Backward-Substitution**.

\[ 2x + 4y - 2z = 2 \]
\[ y + z = 4 \]
\[ 4z = 8 \]

- substitute \( z \) in the first two equations
- after that substitute \( y \) in the first equation
**Parallelization Strategies**

**Blocking:** Matrix-Matrix operations

**Data Mapping:** Load balancing

Example: 4 processors

---

**HPLinpack Characteristics**

- Is most suitable for loosely coupled systems, i.e. lots of low-performance CPUs connected with a relatively low-speed network.
- Is not suitable for SMPs as MPI incurs overhead which causes substantial deterioration of performance for a benchmark code.
- When look-ahead technique is used with MPI, it requires additional memory to be allocated on each CPU for communication buffer. In an SMP system, such buffer is unnecessary due to the shared memory mechanism.
- Optimized BLAS routines can be used to improve CPU performance.
Exercises

- **HPLinpack Benchmark**
  - (Compile and) run the MPI version of the High Performance Linpack
  - Analyze the benchmark performance of OCuLUS and Noctua system

---

**High Performance Conjugate Gradients - HPCG**

- HPCG uses a simple implementation of the Conjugate Gradients and multigrid algorithms
- Performance can be impacted by many system capabilities
  - Sparse matrix-vector multiplication.
  - Global dot products.
  - Local symmetric Gauss-Seidel smoother with sparse triangular solve
- Strictly synchronous implementations, no latency hiding for iterative methods apply
- Represents a large class of applications
  - Demands lot from a computer system
  - Lower bound on achievable performance
- Reference implementation available
  - written in C++ with MPI and OpenMP support
The CG Method

- Minimize the quadratic function \( Q: \mathbb{R}^n \to \mathbb{R} \) given by
  \[
  Q(x) = \frac{1}{2} x^T A x - x^T b
  \]
- Minimum is obtained by setting the gradient equal to zero
  \[
  \nabla Q(x) = A x - b = 0 \quad \text{linear system } A x = b
  \]
- Finding the solution by solving \( r = b - A x = 0 \)
- Sequence \( \{x_k\} \) is such that \( \{r_k\} := \{b - A x_k\} \) is orthogonal with respect to the usual inner product in \( \mathbb{R}^n \)
- The search directions are also orthogonal, but with respect to a different inner product

The CG Algorithm

Start with some \( x_0 \). Set \( p_0 = r_0 = b - A x_0 \)

For \( k = 0,1,2, \ldots \)

\[
\begin{align*}
x_{k+1} &= x_k + \alpha_k p_k, & \alpha_k &= \frac{r_k^T r_k}{p_k^T A p_k} \\
r_{k+1} &= b - A x_{k+1} = r_k - \alpha_k A p_k \\
p_{k+1} &= r_{k+1} + \beta_k p_k, & \beta_k &= \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}
\end{align*}
\]
Example

\[
\begin{pmatrix}
2 & -1 \\
-1 & 2
\end{pmatrix}
\begin{pmatrix}
x_1 \\
x_2
\end{pmatrix}
= 
\begin{pmatrix}
1 \\
0
\end{pmatrix}
\]

• Start with \( x_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \) and \( p_0 = r_0 = b = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \)

\[
\alpha_0 = \frac{r_0^T r_0}{p_0^T A p_0} = \frac{1}{2}, \quad x_1 = x_0 + \alpha_0 p_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix} + \frac{1}{2} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1/2 \\ 0 \end{pmatrix}
\]

• \( r_1 = r_0 - \alpha_0 A p_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} - \frac{1}{2} \begin{pmatrix} 2 \\ -1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1/2 \end{pmatrix} \), \( r_1^T r_0 = 0 \)

\[
\beta_0 = \frac{r_1^T r_1}{r_0^T r_0} = \frac{1}{4}, \quad p_1 = r_1 + \beta_0 p_0 = \begin{pmatrix} 0 \\ 1/2 \end{pmatrix} + \frac{1}{4} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1/4 \\ 1/2 \end{pmatrix}
\]

\[
\alpha_1 = \frac{r_1^T r_1}{p_1^T A p_1} = \frac{2}{3}, \quad x_2 = x_1 + \alpha_1 p_1 = \begin{pmatrix} 1/2 \\ 0 \end{pmatrix} + \frac{2}{3} \begin{pmatrix} 1/4 \\ 1/2 \end{pmatrix} = \begin{pmatrix} 2/3 \\ 1/3 \end{pmatrix}
\]

• \( r_2 = \begin{pmatrix} 0 \\ 1/2 \end{pmatrix} - \frac{2}{3} \begin{pmatrix} 2 \\ -1 \end{pmatrix} \begin{pmatrix} 1/4 \\ 1/2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \), \( \Rightarrow x_2 \) is the exact solution

Iterative vs. Exact Method

• Orthogonality of the residuals implies that \( x_m \) is equal to the solution \( x \) of \( A \cdot x \) for some \( m \leq n \)

• If \( x_k \neq x \ \forall \ k = 0, \ldots, n - 1 \) then \( r_k \neq 0 \ \forall \ k = 0, \ldots, n - 1 \) is an orthogonal basis for \( \mathbb{R}^n \).
  But then \( r_n \in \mathbb{R}^n \) is orthogonal for all vectors in \( \mathbb{R}^n \) so
  \( r_n = 0 \) and hence \( x_n = x \)

• So the CG method finds the exact solution in at most \( n \) iterations

• The convergence analysis shows that \( \| x - x_k \|_A \) typically becomes small quite rapidly and we can stop the iteration with \( k \) much smaller that \( n \)
Multigrid Method

- Multigrid uses a sequence of coarse grids to accelerate the fine grid solution
- MG solvers are optimal with $O(n)$ operations
- Constant solution time as problem size grows in proportion to the number of processors (weak scaling)

Source: Robert D. Falgout, Lawrence Livermore National Laboratory

Multi Grid Parallelization

- Basic communication pattern is nearest neighbor
  - Relaxation, interpolation and other functions are not very complex
- Coarse grid with
  - different neighboring processors
  - Potential of idling processors
HPCG Performance Signatures

- Sparse Matrix Vector Multiplication (SpMV):
  - stresses the memory system by using both streaming memory and indexed reads of memory
  - takes advantage of temporal locality (when computing $y = Ax$, each value of $x$ is typically used about 25 times)
  - requires a "neighborhood collective" where each processor communicates with on average 26 neighboring processors (overlap of communication and local computation of the $y$ values possible)
  - This kernel can be vectorized, either with small vector lengths (order 25) in the reference sparse matrix format used in HPCG, or with long vector lengths (order $N$), if a different (e.g., ELLPACK) format is used.

- Symmetric Gauss-Seidel smoother (SymGS):
  - form of sparse triangular solve requiring fine-grain recursive execution that tests the latency optimization capabilities of a processor (indexed reads)
  - presents a challenging target for fine-grain cooperative threading

HPCG Performance Signatures (cont.)

- Global Dot Product:
  - scanning of a large distributed array of values to produce a single value that is available to every processor on the system
  - requires efficient execution of this operation across all processors

- Vector Update:
  - tests the raw streaming bandwidth of the processor (essentially a STREAMS)
  - consumes a tiny fraction of the overall execution time in HPCG

- Multigrid preconditioner:
  - provides further challenge to the processor by presenting all of the above kernels at 4 different grid sizes
  - grid sizes vary in size by a factor of 8 from level to level
  - coarsest grid is approximately 4,000 times smaller than the original fine grid
  - driven by multigrid preconditioned conjugate gradient algorithm that exercises the key kernels on a nested set of coarse grids.
Exercises

- HPCG Benchmark
  - (Compile and) run the High Performance CG Benchmark
  - Analyze the benchmark performance of the Noctua system