Chapter 3

Multicore and Multiprocessor Systems: Part V

Sparse Linear Systems of Equations

The Conjugate Gradient (CG) Method (a prototype iterative solver)

Algorithm 4: Conjugate Gradient Method

Input: $A \in \mathbb{R}^{n\times n}, b \in \mathbb{R}^n, x_0 \in \mathbb{R}^n$
Output: $x = A^{-1}b$

$p_0 = r_0 = b - Ax_0,\ \ \alpha_0 = ||r_0||_2^2;$
for $m = 0, \ldots, n-1$ **do**
  if $\alpha_m \neq 0$ **then**
    $v_m = Ap_m;$
    $\lambda_m = \frac{\alpha_m}{(v_m, p_m)};$
    $x_{m+1} = x_m + \lambda_m p_m;$
    $r_{m+1} = r_m - \lambda_m v_m;$
    $\alpha_{m+1} = ||r_{m+1}||_2^2;$
    $p_{m+1} = r_{m+1} + \frac{\alpha_{m+1}}{\alpha_m} p_m;$
  else
    STOP;

Sparse Linear Systems of Equations

The Conjugate Gradient (CG) Method (a prototype iterative solver)

Algorithm 4: Conjugate Gradient Method

Input: $A \in \mathbb{R}^{n \times n}, b \in \mathbb{R}^n, x_0 \in \mathbb{R}^n$
Output: $x = A^{-1}b$

$p_0 = r_0 = b - Ax_0,\ \alpha_0 = ||r_0||_2^2$;
for $m = 0, \dots, n-1$ **do**
    if $\alpha_m \neq 0$ **then**
        $v_m = Ap_m$;
        $\lambda_m = \frac{\alpha_m}{(v_m, p_m)}$;
        $x_{m+1} = x_m + \lambda_m p_m$;
        $r_{m+1} = r_m - \lambda_m v_m$;
        $\alpha_{m+1} = ||r_{m+1}||_2^2$;
        $p_{m+1} = r_{m+1} + \frac{\alpha_{m+1}}{\alpha_m} p_m$;
    else
        STOP;

CG uses

- one matrix vector product (performing the main work),

Sparse Linear Systems of Equations

The Conjugate Gradient (CG) Method (a prototype iterative solver)

Algorithm 4: Conjugate Gradient Method

```
Input: A ∈ ℝ^{n×n}, b ∈ ℝ^n, x_0 ∈ ℝ^n
Output: x = A^{-1} b
p_0 = r_0 = b − A x_0,  α_0 = ||r_0||_2^2;
for m = 0, ..., n − 1 do
    if α_m ≠ 0 then
        v_m = A p_m;
        λ_m = α_m / (v_m, p_m);
        x_{m+1} = x_m + λ_m p_m;
        r_{m+1} = r_m − λ_m v_m;
        α_{m+1} = ||r_{m+1}||_2^2;
        p_{m+1} = r_{m+1} + (α_{m+1} / α_m) p_m;
    else
        STOP;
```

CG uses

- one `dot`,

Sparse Linear Systems of Equations

The Conjugate Gradient (CG) Method (a prototype iterative solver)

Algorithm 4: Conjugate Gradient Method

Input: $A \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^n$, $x_0 \in \mathbb{R}^n$
Output: $x = A^{-1}b$

```
p_0 = r_0 = b − A x_0,  α_0 = ||r_0||_2^2;
for m = 0, ..., n − 1 do
    if α_m ≠ 0 then
        v_m = A p_m;
        λ_m = α_m / (v_m, p_m);
        x_{m+1} = x_m + λ_m p_m;
        r_{m+1} = r_m − λ_m v_m;
        α_{m+1} = ||r_{m+1}||_2^2;
        p_{m+1} = r_{m+1} + (α_{m+1} / α_m) p_m;
    else
        STOP;
```

CG uses

- two `axpy`,

Sparse Linear Systems of Equations

The Conjugate Gradient (CG) Method (a prototype iterative solver)

Algorithm 4: Conjugate Gradient Method

Input: $A \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^n$, $x_0 \in \mathbb{R}^n$
Output: $x = A^{-1}b$

```
p_0 = r_0 = b - A x_0,  α_0 = ||r_0||_2^2;
for m = 0, ..., n-1 do
    if α_m ≠ 0 then
        v_m = A p_m;
        λ_m = α_m / (v_m, p_m);
        x_{m+1} = x_m + λ_m p_m;
        r_{m+1} = r_m - λ_m v_m;
        α_{m+1} = ||r_{m+1}||_2^2;
        p_{m+1} = r_{m+1} + (α_{m+1} / α_m) p_m;
    else
        STOP;
```

CG uses

- one `nrm2`,

Sparse Linear Systems of Equations

The Conjugate Gradient (CG) Method (a prototype iterative solver)

Algorithm 4: Conjugate Gradient Method

Input: $A \in \mathbb{R}^{n \times n}, b \in \mathbb{R}^n, x_0 \in \mathbb{R}^n$

Output: $x = A^{-1}b$

$p_0 = r_0 = b - Ax_0,\ \alpha_0 = ||r_0||_2^2;$

for $m = 0, \dots, n-1$ **do**

  if $\alpha_m \neq 0$ **then**

    $v_m = Ap_m;$

    $\lambda_m = \frac{\alpha_m}{(v_m, p_m)};$

    $x_{m+1} = x_m + \lambda_m p_m;$

    $r_{m+1} = r_m - \lambda_m v_m;$

    $\alpha_{m+1} = ||r_{m+1}||_2^2;$

    $p_{m+1} = r_{m+1} + \frac{\alpha_{m+1}}{\alpha_m} p_m;$

  else

    STOP;

CG uses

- and a nonstandard `axpy` operation with result in `x`.

Sparse Linear Systems of Equations

Sparse Matrix Vector Products

The key ingredient in the CG method is the sparse matrix vector product (SpMVP).

We learned in part 1 of the lecture that sparse matrix operations are *bandwidth limited*, i.e., the crucial point is always the data transfer for matrix pattern and entries to the processing units.

On the other hand, the SpMVP is trivially parallel due to data parallelism. On multicore architectures the obvious questions are:

- What is the optimal number of threads to use?
- How should the data be distributed among the threads?

First one: treated in the exercises.

Sparse Linear Systems of Equations

Sparse Matrix Vector Products

The second questions is investigated a lot in the literature. We will only sketch a small selection of approaches considering $x = Ab$ for $x, b \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n\times n}$ sparse with properties specified separately in the method descriptions.

Sparse Linear Systems of Equations

Sparse Matrix Vector Products

The second questions is investigated a lot in the literature. We will only sketch a small selection of approaches considering $x = Ab$ for $x, b \in \mathbb{R}^n$ and $A \in \mathbb{R}^{n \times n}$ sparse with properties specified separately in the method descriptions.

Naive row blocking. (e.g., using OpenMP `parallel for`)

If the matrix A is banded with moderate bandwidth and the number of entries per row is almost the same for all rows, simply grouping the rows in blocks of rows will likely do a good job.

The bandwidth limitations guarantee data locality on b.

Furthermore, the similar lengths of the sparse rows will automatically provide a proper load balancing.

This provides the easiest form of 1d-partitioning.

Sparse Linear Systems of Equations

Sparse Matrix Vector Products

The simplest form of 2d-partitioning of the matrix A uses (blocks of) columns and (blocks of) rows at the same time. It is usually referred to as hypergraph partitioning since the choice fit the following definition.

Definition (Hypergraph)

A *hypergraph* is an ordered pair $(\mathcal{V}, \mathcal{E})$ of sets. It is a generalization of a graph that consists of vertices (in the set $\mathcal{V}$) and *hyperedges* in the set $\mathcal{E}$. In contrast to an edge in a graph a hyperedge can be an arbitrary subset of $\mathcal{V}$ and not just a pair.

Sparse Linear Systems of Equations

Sparse Matrix Vector Products

Example

Schematic representation of a hypergraph with seven vertices and four hyperedges.

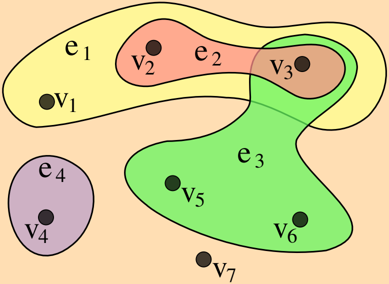


Image source: https://commons.wikimedia.org/wiki/File:Hypergraph-wikipedia.svg

Sparse Linear Systems of Equations

Sparse Matrix Vector Products

The idea of hypergraph partitioning is to use the hyperedges to find the optimal partitioning of the vertices into k equal sets for optimal balancing of the workload and data communication.

The problem of finding the optimal partition is however np-hard. Therefore cheap heuristics are employed to approximate the optimal partition.

An interesting variant especially for symmetric patterns is the corner symmetric partitioning.

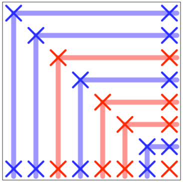


Figure: Corner symmetric partitioning of the arrowhead matrix with 2 partitions.

Sparse Linear Systems of Equations

Sparse Matrix Vector Products

$$\begin{bmatrix} * & & & & & & & * \\ & * & & & & & & * \\ & & * & & & & & * \\ & & & * & & & & * \\ & & & & * & & & * \\ & & & & & * & & * \\ & & & & & & * & * \\ * & * & * & * & * & * & * & * \end{bmatrix}$$

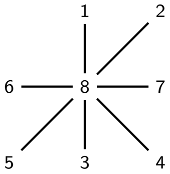


Figure: arrowhead matrix pattern and connectivity graph.

The central node 8 is called *vertex separator*. The identification of such a (group of) node(s) is the central question in the graph model based partitioning. Successive application of this idea leads to the nested dissection scheme.

Sparse Linear Systems of Equations

Sparse Matrix Vector Products

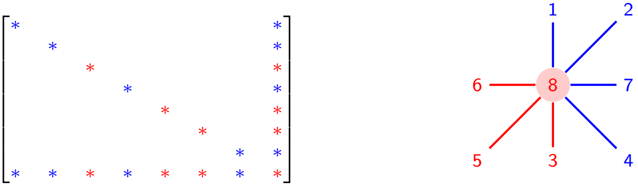


Figure: arrowhead matrix pattern and connectivity graph.

The central node 8 is called *vertex separator*. The identification of such a (group of) node(s) is the central question in the graph model based partitioning. Successive application of this idea leads to the nested dissection scheme.

Sparse Linear Systems of Equations

Preconditioning

Recall:

A *preconditioner* is an invertible linear operator P that approximates the action of A^{-1} for a linear system $Ax = b$.

- Invertibility required to ensure proper preservation of solution,
- preconditioner need not be formed as a matrix, as long as its action on a vector can be provided as a function,
- main purpose of the preconditioner is the grouping of eigenvalues, ideally in a single cluster at $+1$.

Sparse Linear Systems of Equations

Preconditioned CG

Algorithm 5: Preconditioned Conjugate Gradient Method

Input: $A \in \mathbb{R}^{n\times n}$, $b \in \mathbb{R}^n$, $x_0 \in \mathbb{R}^n$, $A^{-1} \approx P \in \mathbb{R}^{n\times n}$
Output: $x = A^{-1}b$

$r_0 = b - Ax_0$, $p_0 = z_0 = Pr_0$, $\alpha_0 = (r_0, p_0)$;
for $m = 0 : n-1$ **do**
  if $\alpha_m \neq 0$ **then**
    $v_m = Ap_m$;
    $\lambda_m = \frac{\alpha_m}{(v_m, p_m)_2}$;
    $x_{m+1} = x_m + \lambda_m p_m$;
    $r_{m+1} = r_m - \lambda_m v_m$;
    $z_{m+1} = Pr_{m+1}$;
    $\alpha_{m+1} = (r_{m+1}, z_{m+1})_2$;
    $p_{m+1} = z_{m+1} + \frac{\alpha_{m+1}}{\alpha_m} p_m$;
  else
    STOP;

Sparse Linear Systems of Equations

Diagonal/Jacobi Preconditioner

Let $D \in \mathbb{R}^{n \times n}$ be a diagonal matrix containing the diagonal of A. Then $P = D^{-1}$ is called Jacobi or diagonal preconditioner.

Properties

- $+$ embarrassingly parallel in computation and application,
- $+$ storage requirement n `double` numbers,
- $-$ only useful for diagonally dominant systems.

Sparse Linear Systems of Equations

Sparse Approximate Inverse (SPAI) Preconditioning

The basic idea of SPAI is to find the best matrix P approximating A^{-1}, while maintaining the sparsity pattern of A.

$$\min_{\mathcal{P}(P)=\mathcal{P}(A)} ||AP - I||_F^2 = \min_{\mathcal{P}(P)=\mathcal{P}(A)} \underbrace{\sum_{j=1}^{n} ||Ap_j - e_j||_F^2}_{n \text{ independent least squares problems}}$$

Sparse Linear Systems of Equations

Sparse Approximate Inverse (SPAI) Preconditioning

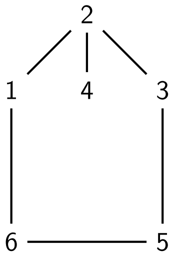
The basic idea of SPAI is to find the best matrix P approximating A^{-1}, while maintaining the sparsity pattern of A.

$$\min_{\mathcal{P}(P)=\mathcal{P}(A)} ||AP - I||_F^2 = \min_{\mathcal{P}(P)=\mathcal{P}(A)} \underbrace{\sum_{j=1}^{n} ||Ap_j - e_j||_F^2}_{n \text{ independent least squares problems}}$$

- $+$ only SpMVP needed for the application,
- $+$ n independent least squares problems allow two multicore approaches:
  - rely on threaded BLAS when solving the least squares problems sequentially via `dgeqrs()` from LAPACK,
  - use sequential BLAS with OpenMP for parallel solution of the least squares problems.
- $-$ efficient preconditioning requires additional fill-in, which leads to extra storage and computation complexity.

Sparse Linear Systems of Equations

Issues of Sparse Direct Solvers



(a) initial graph $\mathcal{G}_0$

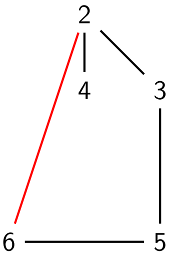
$$H_0 = \begin{bmatrix} 1 & * & & & & * \\ * & 2 & * & * & & \\ & * & 3 & & * & \\ & * & & 4 & & \\ & & * & & 5 & * \\ * & & & & * & 6 \end{bmatrix}$$

(b) corresponding submatrix 0

Figure: Basic graph elimination procedure for a symmetric matrix and the Cholesky decomposition

Sparse Linear Systems of Equations

Issues of Sparse Direct Solvers



(c) elimination graph $\mathcal{G}_1$

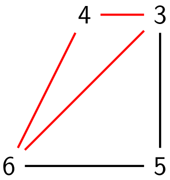
$$H_1 = \begin{bmatrix} 2 & * & * & & {\color{red}*} \\ * & 3 & & * & \\ * & & 4 & & \\ & * & & 5 & * \\ {\color{red}*} & & & * & 6 \end{bmatrix}$$

(d) corresponding submatrix 1

Figure: Basic graph elimination procedure for a symmetric matrix and the Cholesky decomposition

Sparse Linear Systems of Equations

Issues of Sparse Direct Solvers



(e) elimination graph $\mathcal{G}_2$

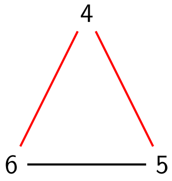
$$H_2 = \begin{bmatrix} 3 & * & * & * \\ * & 4 & & * \\ * & & 5 & * \\ * & * & * & 6 \end{bmatrix}$$

(f) corresponding submatrix 2

Figure: Basic graph elimination procedure for a symmetric matrix and the Cholesky decomposition

Sparse Linear Systems of Equations

Issues of Sparse Direct Solvers



(g) elimination graph $\mathcal{G}_3$

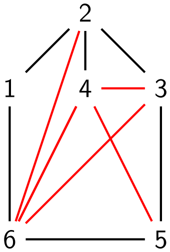
$$H_3 = \begin{bmatrix} 4 & * & * \\ * & 5 & * \\ * & * & 6 \end{bmatrix}$$

(h) corresponding submatrix 3

Figure: Basic graph elimination procedure for a symmetric matrix and the Cholesky decomposition

Sparse Linear Systems of Equations

Issues of Sparse Direct Solvers



$$F = \begin{bmatrix} 1 & * & & & & * \\ * & 2 & * & * & & * \\ & * & 3 & * & * & * \\ & * & * & 4 & * & * \\ & & * & * & 5 & * \\ * & * & * & * & * & 6 \end{bmatrix}$$

(a) The filled graph $\mathcal{G}^+(A) = \mathcal{G}(F)$ (b) The final matrix $F = L + L^T$ with fill.

Figure: The filled graph and matrix of a Cholesky decomposition example.

Sparse Linear Systems of Equations

Issues of Sparse Direct Solvers

Now

$$L = \begin{bmatrix} 1 & & & & & \\ * & 2 & & & & \\ & * & 3 & & & \\ & * & * & 4 & & \\ & & * & * & 5 & \\ * & * & * & * & * & 6 \end{bmatrix}$$

and thus, the forward elimination is purely sequential. Are we lost?

Sparse Linear Systems of Equations

Issues of Sparse Direct Solvers

Consider the Cholesky factor:

$$L = \begin{bmatrix} 1 & & & & & & & & & \\ * & 2 & & & & & & & & \\ & & 3 & & & & & & & \\ & & * & 4 & & & & & & \\ & & * & * & 5 & & & & & \\ & & & & & 6 & & & & \\ & & * & * & * & & 7 & & & \\ & * & * & * & * & & * & 8 & & \\ & & & & & * & & & 9 & \\ & * & * & * & * & & * & * & * & 10 \end{bmatrix}$$

Sparse Linear Systems of Equations

Issues of Sparse Direct Solvers

Consider the Cholesky factor:

$$L = \begin{bmatrix} 1 &&&&&&&&& \\ * & 2 &&&&&&&& \\ && 3 &&&&&&& \\ && * & 4 &&&&&& \\ && * & * & 5 &&&&& \\ &&&&& 6 &&&& \\ && * & * & * && 7 &&& \\ & * & * & * & * && * & 8 && \\ &&&&& * &&& 9 & \\ & * & * & * & * && * & * & * & 10 \end{bmatrix}$$

Sparse Linear Systems of Equations

Issues of Sparse Direct Solvers

Consider the Cholesky factor:

$$L = \begin{bmatrix} 1 & & & & & & & & & \\ * & 2 & & & & & & & & \\ & & 3 & & & & & & & \\ & & * & 4 & & & & & & \\ & & * & * & 5 & & & & & \\ & & & & & 6 & & & & \\ & & * & * & * & & 7 & & & \\ & * & * & * & * & & * & 8 & & \\ & & & & & * & & & 9 & \\ & * & * & * & * & & * & * & * & 10 \end{bmatrix}$$

Sparse Linear Systems of Equations

Issues of Sparse Direct Solvers

Definition (column pattern)

The j-th *column pattern* $\mathcal{P}_{*j}$ is the set of row indices of all non-diagonal nonzero entries in the j-th column.

Definition (Supernode)

A *supernode* is a set of contiguous column indices

$$\mathcal{I}(p) = \{p, p+1, \dots, p+q-1\},$$

such that for all columns $i \in \mathcal{I}(p)$ we have

$$\mathcal{P}_{*i} = \mathcal{P}_{*(p+q-1)} \cup \{i+1, \dot{;} p+q-1\}$$

Sparse Linear Systems of Equations

Issues of Sparse Direct Solvers

- Supernodes, thus are special dense diagonal blocks that have the identically same pattern in each column below the diagonal block.

Sparse Linear Systems of Equations

Issues of Sparse Direct Solvers

- Supernodes, thus are special dense diagonal blocks that have the identically same pattern in each column below the diagonal block.
- Column modifications in forward substitution can be expressed in terms of supernodes rather than single diagonal entries.

Sparse Linear Systems of Equations

Issues of Sparse Direct Solvers

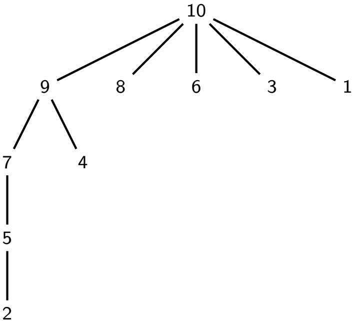
- Supernodes, thus are special dense diagonal blocks that have the identically same pattern in each column below the diagonal block.
- Column modifications in forward substitution can be expressed in terms of supernodes rather than single diagonal entries.
- Inside the supernode block operations we can exploit parallelism.

Sparse Linear Systems of Equations

A Task Pool Approach to Parallel Triangular Solves

Consider the Cholesky factor and corresponding elimination tree

$$L = \begin{bmatrix} 1 & & & & & & & & & \\ & 2 & & & & & & & & \\ & & 3 & & & & & & & \\ & & & 4 & & & & & & \\ & * & & & 5 & & & & & \\ & & & & & 6 & & & & \\ & & & & * & & 7 & & & \\ & & & & & & & 8 & & \\ & * & & * & * & & * & & 9 & \\ * & * & * & * & * & * & * & * & * & 10 \end{bmatrix}$$



Sparse Linear Systems of Equations

A Task Pool Approach to Parallel Triangular Solves

+ many elimination steps can be executed independently

Sparse Linear Systems of Equations

A Task Pool Approach to Parallel Triangular Solves

+ many elimination steps can be executed independently

+ a simple task pool scheduling the independent tasks enables parallel execution and load balancing

Sparse Linear Systems of Equations

A Task Pool Approach to Parallel Triangular Solves

+ many elimination steps can be executed independently
+ a simple task pool scheduling the independent tasks enables parallel execution and load balancing
− elimination tree must be computed to enable proper scheduling and identification of independent tasks

Relevant Software and Libraries

Dense Linear Algebra

1. **OpenBLAS** based on the earlier GotoBLAS project OpenBLAS implements a complete set of optimized BLAS routines. On a machine with a single socket it is likely the fastest BLAS implementation one can get. [a]

---

[a] http://xianyi.github.io/OpenBLAS/
[b] http://software.intel.com/en-us/intel-mkl
[c] http://icl.cs.utk.edu/plasma/software/

Relevant Software and Libraries

Dense Linear Algebra

1. **OpenBLAS** based on the earlier GotoBLAS project OpenBLAS implements a complete set of optimized BLAS routines. On a machine with a single socket it is likely the fastest BLAS implementation one can get. [a]
2. **Intel® Math Kernel Library (MKL)** is Intel®s optimized implementation of BLAS and LAPACK. It is the strongest opponent of OpenBLAS on single socket systems. On a system with several sockets no other BLAS library outperforms MKL. [b]

[a] http://xianyi.github.io/OpenBLAS/
[b] http://software.intel.com/en-us/intel-mkl
[c] http://icl.cs.utk.edu/plasma/software/

Relevant Software and Libraries

Dense Linear Algebra

1. **OpenBLAS** based on the earlier GotoBLAS project OpenBLAS implements a complete set of optimized BLAS routines. On a machine with a single socket it is likely the fastest BLAS implementation one can get. [a]
2. **Intel® Math Kernel Library (MKL)** is Intel®s optimized implementation of BLAS and LAPACK. It is the strongest opponent of OpenBLAS on single socket systems. On a system with several sockets no other BLAS library outperforms MKL. [b]
3. **PLASMA** The **P**arallel **L**inear **A**lgebra **S**ubroutines for **M**ulticore **A**rchitectures employs DAG scheduling to increase performance of the linear algebra subsystem on multicore architectures. [c]

[a] http://xianyi.github.io/OpenBLAS/
[b] http://software.intel.com/en-us/intel-mkl
[c] http://icl.cs.utk.edu/plasma/software/

Relevant Software and Libraries

Sparse Linear Algebra

1. **UMFPACK** comes as part of the SuiteSparse package of software libraries for sparse linear systems of equations. Uses thread parallel multifrontal techniques to solve linear systems of equations. [a]

---

[a] http://www.cise.ufl.edu/research/sparse/umfpack/
[b] http://www.boost.org/doc/libs/1_53_0/libs/numeric/ublas/doc/index.htm
[c] http://www.simunova.com/en/node/24
[d] http://crd-legacy.lbl.gov/~xiaoye/SuperLU/

Relevant Software and Libraries

Sparse Linear Algebra

1. **UMFPACK** comes as part of the SuiteSparse package of software libraries for sparse linear systems of equations. Uses thread parallel multifrontal techniques to solve linear systems of equations. [a]
2. **Boost uBLAS** *"is a C++ template class library that provides BLAS level 1, 2, 3 functionality for dense, packed and sparse matrices."*[b]

[a] http://www.cise.ufl.edu/research/sparse/umfpack/
[b] http://www.boost.org/doc/libs/1_53_0/libs/numeric/ublas/doc/index.htm
[c] http://www.simunova.com/en/node/24
[d] http://crd-legacy.lbl.gov/~xiaoye/SuperLU/

Relevant Software and Libraries

Sparse Linear Algebra

1. **UMFPACK** comes as part of the SuiteSparse package of software libraries for sparse linear systems of equations. Uses thread parallel multifrontal techniques to solve linear systems of equations. [a]
2. **Boost uBLAS** *"is a C++ template class library that provides BLAS level 1, 2, 3 functionality for dense, packed and sparse matrices."*[b]
3. **MTL** the **M**atrix **T**emplate Łibrary provides an easy to use template based C++ interface to linear algebra operations. It relies on Boost for fast and efficient codes.
[c]

---

[a] http://www.cise.ufl.edu/research/sparse/umfpack/
[b] http://www.boost.org/doc/libs/1_53_0/libs/numeric/ublas/doc/index.htm
[c] http://www.simunova.com/en/node/24
[d] http://crd-legacy.lbl.gov/~xiaoye/SuperLU/

Relevant Software and Libraries

Sparse Linear Algebra

1. **UMFPACK** comes as part of the SuiteSparse package of software libraries for sparse linear systems of equations. Uses thread parallel multifrontal techniques to solve linear systems of equations. [a]
2. **Boost uBLAS** *"is a C++ template class library that provides BLAS level 1, 2, 3 functionality for dense, packed and sparse matrices."*[b]
3. **MTL** the **M**atrix **T**emplate Łibrary provides an easy to use template based C++ interface to linear algebra operations. It relies on Boost for fast and efficient codes.
[c]
4. **SuperLU_MT** Supernode based multithreaded LU decomposition.[d]

[a] http://www.cise.ufl.edu/research/sparse/umfpack/
[b] http://www.boost.org/doc/libs/1_53_0/libs/numeric/ublas/doc/index.htm
[c] http://www.simunova.com/en/node/24
[d] http://crd-legacy.lbl.gov/~xiaoye/SuperLU/

Relevant Software and Libraries

PThreads and Scheduling/Memory Control

1. **nptl** is the **N**ative **P**OSIX **L**inux **T**hread library that currently provides PThread support on most Linux platforms. [a]

[a] http://en.wikipedia.org/wiki/Native_POSIX_Thread_Library
[b] http://code.google.com/p/likwid/
[c] http://oss.sgi.com/projects/libnuma/

Relevant Software and Libraries

PThreads and Scheduling/Memory Control

1. **nptl** is the **N**ative **P**OSIX **L**inux **T**hread library that currently provides PThread support on most Linux platforms. [a]
2. **likwid** (**L**ike **I** **K**new **W**hat **I** **D**o) is a light weight library that supports software developers to design high performance scientific computing programs with little overhead. [b]

---

[a] http://en.wikipedia.org/wiki/Native_POSIX_Thread_Library
[b] http://code.google.com/p/likwid/
[c] http://oss.sgi.com/projects/libnuma/

Relevant Software and Libraries

PThreads and Scheduling/Memory Control

1. **nptl** is the **N**ative **P**OSIX **L**inux **T**hread library that currently provides PThread support on most Linux platforms. [a]
2. **likwid** (**L**ike **I** **K**new **W**hat **I** **D**o) is a light weight library that supports software developers to design high performance scientific computing programs with little overhead. [b]
3. **numactl** referred to as libnuma by several Linux distributions, numactl is a small program/library that can be used to control placement of process memory in NUMA environments. The library version seems to be preferred by the Linux kernel policies. [c]

[a] http://en.wikipedia.org/wiki/Native_POSIX_Thread_Library
[b] http://code.google.com/p/likwid/
[c] http://oss.sgi.com/projects/libnuma/