Concepts and Models of Parallel and Data-centric Programming

Parallel Algorithms III

Lecture, Summer 2020

Dr. Christian Terboven <terboven@itc.rwth-aachen.de>

i12 High Performance Computing
RWTH AACHEN UNIVERSITY

Outline

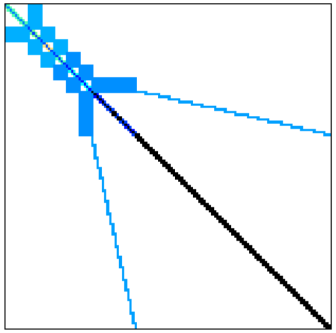
0. Organization
1. Foundations
2. Shared Memory
3. GPU Programming
4. Bulk-Synchronous Parallelism
5. Message Passing
6. Distributed Shared Memory
7. **Parallel Algorithms**
8. Parallel I/O
9. MapReduce
10. Apache Spark

a. Berkeley DWARFS
b. Dense Linear Algebra
c. Sparse Linear Algebra
d. Monte Carlo Methods
e. Graph Traversal

Sparse Linear Algebra

Motivation

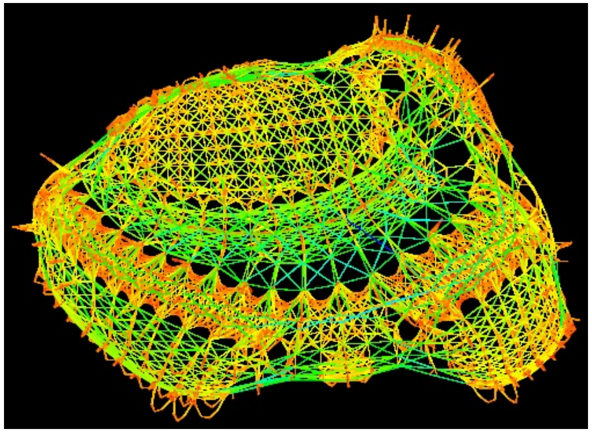
- Sparse Linear Algebra
  - Sparse Linear Equation Systems occur in many scientific disciplines.
  - Sparse matrix-vector multiplications (SpMxV) are the dominant part in many iterative solvers (like CG) for such systems.
  - number of non-zeros << n*n



Beijing Botanical Garden

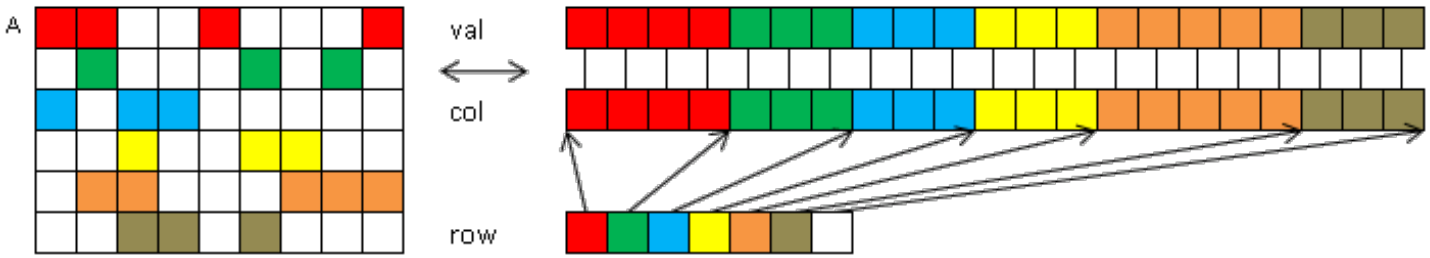
Upper right: Original building
Lower right: Model
Lower left: Matrix

(Source: Beijing Botanical Garden and University of Florida, Sparse Matrix Collection)



Representation of a sparse matrix

- CRS: Compressed Row Storage



- A concrete example

$$A = \begin{pmatrix} \underset{(0,0)}{10} & 0 & 0 & \underset{(0,3)}{12} & 0 \\ 0 & 0 & \underset{(1,2)}{11} & 0 & \underset{(1,4)}{13} \\ 0 & \underset{(2,1)}{16} & 0 & 0 & 0 \\ 0 & 0 & \underset{(3,2)}{11} & 0 & \underset{(3,4)}{13} \end{pmatrix}$$

$$val = \begin{pmatrix} \underset{(0,0)}{10} & \underset{(0,3)}{12} & \underset{(1,2)}{11} & \underset{(1,4)}{13} & \underset{(2,1)}{16} & \underset{(3,2)}{11} & \underset{(3,4)}{13} \end{pmatrix}$$

$$colInd = \begin{pmatrix} \underset{(0}{0} & \underset{)}{3} & \underset{(1}{2} & \underset{)}{4} & \underset{(2)}{1} & \underset{(3}{2} & \underset{)}{4} \end{pmatrix}$$

$$rowPtr = \begin{pmatrix} \underset{(0)}{0} & \underset{(1)}{2} & \underset{(2)}{4} & \underset{(3)}{5} & \underset{(4)}{7} \end{pmatrix}$$

Case Study CG / 1

Hotspot analysis of the serial code:

Call Stack	CPU Time: Total by Utilization (Idle, Poor, Ok, Ideal, Over)	
cg	46.7%	1.
matvec	40.8%	2.
xpay	1.4%	2.
axpy	1.4%	3.
vectorDot	1.2%	2.
axpy	1.1%	3.
vectorDot	0.6%	

Hotspots are:

1. matrix-vector multiplication
2. scaled vector additions
3. dot product

SpMXV

- SpMXV: sparse matrix vector multiplication

- Algorithm (y = A * x) adopted for the CRS format:

```
for i = 1, n
  y(i) = 0
  for j = row(i), row(i+1) - 1
    y(i) = y(i) + val(j) * x(col(j))
  end;
end;
```

- What is the intuitive approach to parallelize such a loop?

Case Study CG / 2

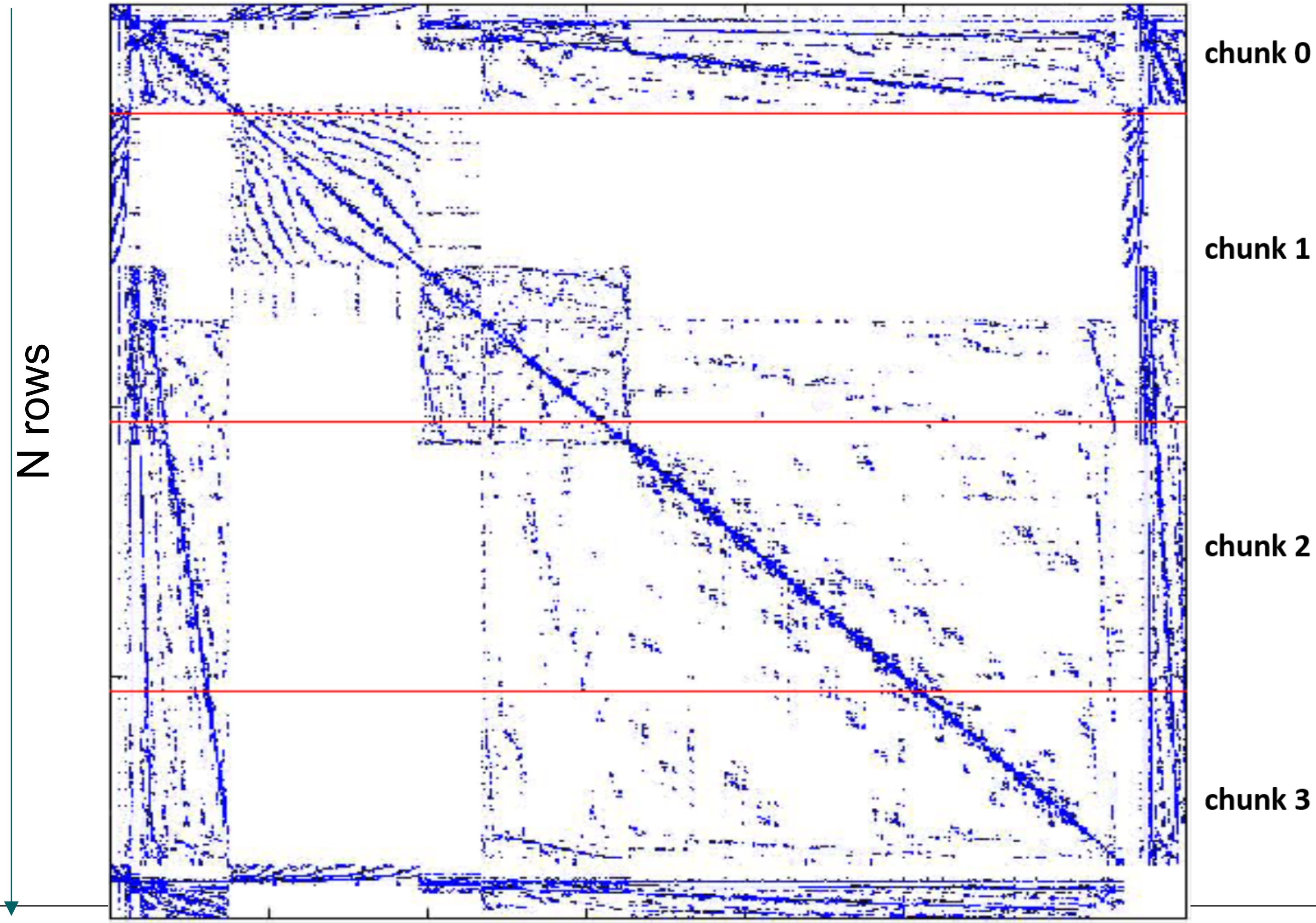
- Analyzing load imbalance in the concurrency view:

So.. Line	Source	CPU Time: Total by… Idle Poor Ok Ic		Ove… and…
49	`void matvec(const int n, const int nnz, .`			
50	`int i,j;`			
51	`#pragma omp parallel for private(j)`	22.462s		10.612s
52	`for(i=0; i<n; i++){`	0.050s		0s
53	`y[i]=0;`	0.060s		0s
54	`for(j=ptr[i]; j<ptr[i+1]; j++){`	1.741s		0s
55	`y[i]+=value[j]*x[index[j]];`	9.998s		0s

- 10 seconds out of ~35 seconds are overhead time
- other parallel regions which are called the same amount of time only produce 1 second of overhead

Matrix structure (spy plot)

- Chunking illustrated for four threads



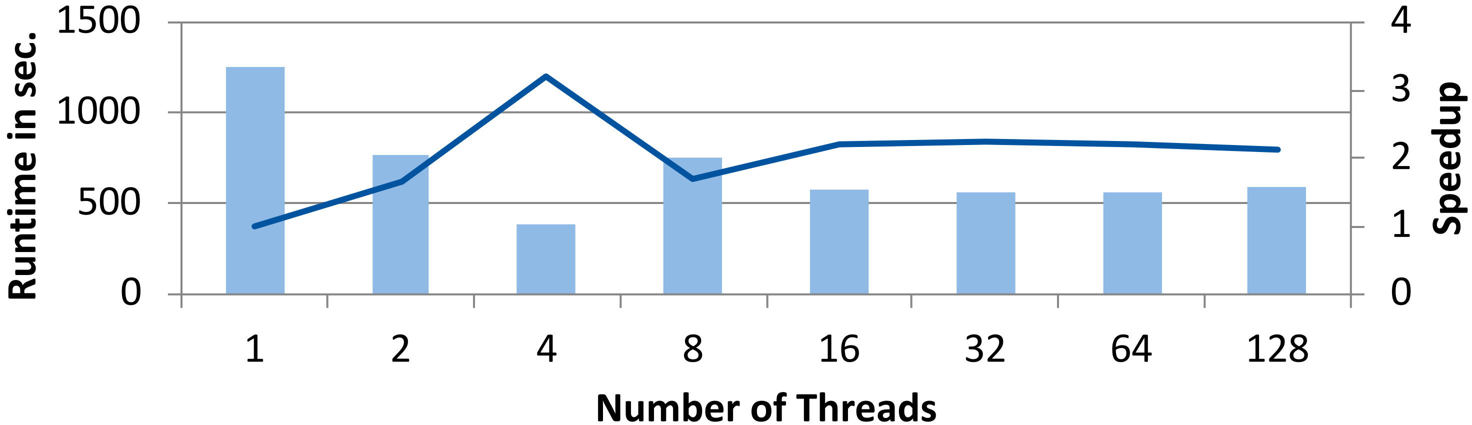
Solution in OpenMP

- Manual computation of the work distribution for load balancing
- Exploitation of the load balancing in the computation:

```
#pragma omp parallel private(i) num_threads(tOptions->iNumThreads)
{
   for (i = start; i < end; i++) {
    double sum = 0.0;
    int rowbeg = Arow[i]; int rowend = Arow[i+1];
    int nz;
    #pragma omp simd reduction(+:sum)
       for (nz = rowbeg; nz < rowend; nz++) {
          sum += Aval[nz] * x[ Acol[nz] ];
       }
        y[i] = sum;
} }
```

SpMXV within CG

- Naive approach:



- Optimal approach: