HIGH PERFORMANCE NUMERICAL LINEAR ALGEBRA

Chao Yang
Computational Research Division
Lawrence Berkeley National Laboratory
Berkeley, CA, USA

BERKELEY LAB
Lawrence Berkeley National Laboratory

BLAS

- BLAS 1, 2, 3
- Performance
- GEMM
- Optimized BLAS
- Parallel BLAS

Three levels of BLAS

- Level-1:
  - AXPY: $y \leftarrow y + \alpha x$
  - DOT: $x^T y$
  - SCAL: $y \leftarrow \alpha y$
  - NRM2: $\|x\|$
- Level-2:
  - GEMV: $y \leftarrow y + \alpha A x$
- Level-3:
  - GEMM: $C \leftarrow C + \alpha A B$

Op count vs memory movement

- AXPY: $y \leftarrow \alpha x + y$
  - Ops: N mults and N adds
  - Loads 2N+1
  - Stores N
  - Ops/data ratio O(1)
- GEMV: $y \leftarrow \alpha A x + \beta y$
  - Ops: MN mults/add
  - Loads MN + N + M
  - Stores M
  - Ops/data ratio: O(1)
- GEMM $C \leftarrow \alpha AB + \beta C$
  - Ops: MNK
  - Loads MN + NK
  - Stores N
  - **Ops/data ratio O(N)**

BLAS1 Vectorization

- AXPY loop

```
For I = 1:n
      Y(i) = alpha*x(i)+y(i)
end
```

- Unrolled loop

```
For I = 1:4:n
      Y(i) = alpha*x(i)+y(i)
      Y(i+1)=alpha*x(i+1)+y(i+1)
      Y(i+2)=alpha*x(i+2)+y(i+2)
      Y(i+3)=alpha*x(i+3)+y(i+3)
end
```

- Vectorized loop

```
For I = 1:4:n
      <load y(i), y(i+1), y(i+2), y(i+3) into a vector register vy>
      <load x(i), x(i+1), x(i+2), x(i+3) into a vector register vx
      vy = alpha*vx+vy
      <store vy into y(i), y(i+1), y(i+2), y(i+3)>
end
```

Loop unrolling for BLAS 2

- Simple implementation

```
for j =1 :n
    for i = 1:n
         y(i) = alpha*A(i,j)*x(j) + beta*y(i)
    end
end
```

- Loop unrolling to reduce the number of writes

```
for j =1 :3:n
    for i = 1:n
         y(i)=A(i,j)*x(j)+A(I,j+1)*x(j+1)
             +A(I,j+2)*x(j+2)+y(i)
    end
end
```

Matrix blocking and tuning GEMM

- C=A·B
- Break A, B and C into blocks of size b
- Block multiplication

```
for j = 1:b:n
    for i = 1:b:n
        for k=1:b:n
            C(i:i+b-1,j:j+b-1)
            =A(i:i+b-1,k:k+b-1)·B(k:k+b-1,j:j+b-1)
        end
    end
end
```

- Number of reads/writes: $\left(\frac{n}{b}\right)^3 4b^2 = 4n^3/b$, minimized when $3b^2$ matrix elements fit in cache (with size M). Lower bound for data movement $\frac{n^3}{\sqrt{M}}$

Recursive GEMM

- Partition A, B, C into 2x2 blocks

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix}$$

- Recursive procedure: C=matmul(A,B,n)

```
function C=matmul(A,B,n)
If (n == 1)
   C=A*B;
else
   C11=matmul(A11,B11,n/2)+matmul(A12,B21,n/2);
   C12=matmul(A11,B12,n/2)+matmul(A12,B22,n/2);
   C21=matmul(A21*B11,n/2)+matmul(A22,B21,n/2);
   C22=matmul(A21,B12,n/2)+matmul(A22,B22,n/2);
endif
```

Cost analysis for Recursive GEMM

- Number of arithmetic operations $A(n)$:

$$A(n) = 8A\left(\frac{n}{2}\right) + 4\left(\frac{n}{2}\right)^2 \text{ if } n > 1;\ A(1) = 1$$

Sum up: $A(n) = 2n^3$

- Number of words moved between fast and slow memory:

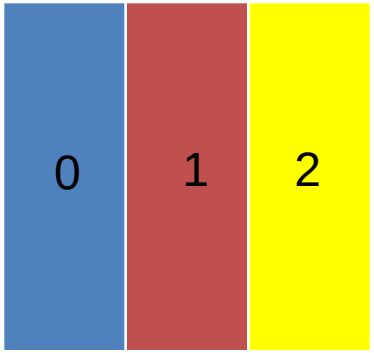
$$O\left(\frac{n^3}{\sqrt{M}} + n^2\right)$$

Tuning the block size

- Performance model depends on many machine characteristics, cache size, cache line size, latency, bandwidth, memory conflict etc.
- Blocking for register, multiple levels of cache, translation lookside buffer (TLB)
- Combine with compiler optimization: instruction pipelining, data prefetch
- Automatic tunning, search for the optimal blocksize at runtime for a particular machine (one-time installation and testing cost)
  - ATLAS www.netlib.org/atlas
  - BeBOP bebop.cs.berkeley.edu
  - PHiPAC www.icsi.berkeley/~bilmes/phipac
- Vendor library: CRAY libsci, AMD acml, Intel mkl, Nvidia CuBLAS

Parallel BLAS

- Data decomposition (ignore cache blocking for the moment)



- Two versions of GEMM $C = AB$
  - Inner product

$$C(i,j) = A(i,:)B(:,j)$$

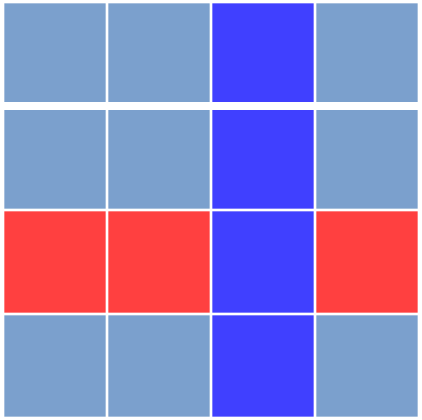
  - Outer product

$$C = \sum_{k=1}^{n} A(:,k)B(k,:)$$

Communication vs computation in GEMM

- Use outer product formulation: $C(i,j) = \sum_{k=1}^{n} A(i,k)B(k,j)$
- SUMMA Algorithm: assume A, B, C 2D-distributed on P processors

```
for k = 1:n
    for i = 1:√p
        Owner of A(i,k) broadcast to other processors
        in the same row
    end;
    for j = 1 :√p
        Owner of A(k,j) broadcast to other processors
        in the same column
    end;
    Receive A(i,k) into Acol
    Receive B(i,j) into Brow
    C(i,j)=C(i,j)+Acol*Brow
end
```



Cost analysis

- Communication cost: latency + #words*time_per_word
- Broadcast is typically implemented by a tree algorithm that requires sending log(p) messages
- Parallel efficiency: serial time/(P*parallel wall time)
- Cost of SUMMA GEMM: assume block size $b = n/p$
  - Total #words communicated: data received by each processor times the number of outer products = $\Omega\left(\frac{n^2}{p} \cdot \frac{n}{b}\right) = \Omega\left(n^2/\sqrt{p}\right)$
  - # of messages: $\Omega(\log(\sqrt{p}))$ or $\Omega(\sqrt{p})$ depending on whether tree algorithm is used
  - Flops: $2n^2b/P = 2n^3/P^{\frac{3}{2}}$

2.5D GEMM algorithm

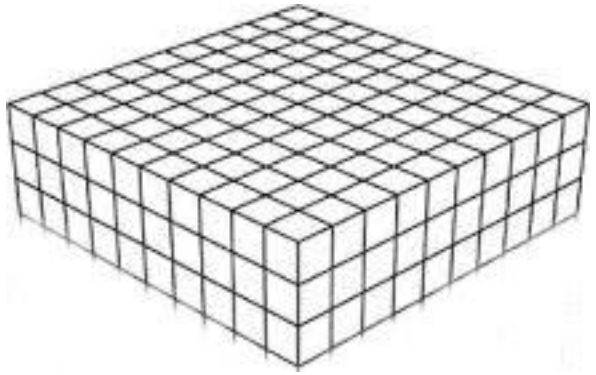
- Replicate each matrix c times (require cn^2 storage)
- Distribute among p processors organized into a 3D processor grid

$$p = \frac{p}{c} \times c = \sqrt{\frac{p}{c}} \times \sqrt{\frac{p}{c}} \times c$$

- Processor $p(i,j,0)$ owns matrix block $A(i,j)$ and $B(i,j)$, each of size $\frac{n}{\sqrt{\frac{p}{c}}} \times \frac{n}{\sqrt{\frac{p}{c}}}$

Algorithm:

1. Processor $p(i,j,0)$ broadcasts $A(i,j)$ and $B(i,j)$ to $p(i,j,k)$ for all k
2. Processors at level k perform $\frac{1}{c}$-th of SUMMA, i.e. 1/c-th of $\sum_m A(i,m)B(m,j)$
3. Reduce partial sums $\sum_m A(i,m)B(m,j)$ along k-axis so $p(i,j,0)$ owns $C(i,j)$



Cost analysis

- Communication volume: $O(\frac{n^2}{\sqrt{cp}})$
- Latency (number of messages): $O(\frac{\sqrt{p}}{\sqrt{c^3}} + \log c)$
- Each processor hold $M = cn^2/p$ data items (doubles)
- Compare with 2D SUMMA:
  - ✓Communication volume: $O(\frac{n^2}{\sqrt{p}})$
  - ✓Latency: $O(\sqrt{p})$

Lots of details in
EuroPar'11 (Solomonik, D.), SC'11 paper by Solomonik, Bhatele, D.
LAWN238, LAWN248

Communication lower bounds

Provably the best you can do:

- Assume M is the data hold by each processor
- Number of words moved/processor:

$$\Omega(\text{flops per processor } / M^{\frac{1}{2}})$$

- Number of messages sent:

$$\Omega(\text{flops per processor } / M^{\frac{3}{2}})$$

2D SUMMA

- bandwidth: $\Omega\left(\frac{\frac{n^3}{p}}{\sqrt{\frac{n^2}{p}}}\right) = \Omega\left(\frac{n^2}{p^{1/2}}\right)$
- Latency:$\Omega\left(\frac{\frac{n^3}{p}}{\left(\frac{n^2}{p}\right)^{3/2}}\right) = \Omega(p^{\frac{1}{2}})$

2.5D SUMMA

- bandwidth: $\Omega\left(\frac{\frac{n^3}{p}}{\sqrt{\frac{cn^2}{p}}}\right) = \Omega\left(\frac{n^2}{\sqrt{cp}}\right)$
- Latency:$\Omega\left(\frac{\frac{n^3}{p}}{\left(\frac{cn^2}{p}\right)^{3/2}}\right) = \Omega(\frac{\sqrt{p}}{\sqrt{c^3}})$