Parallel Algorithms For Dense Linear Algebra Computations

K.A. GALLIVAN , R.J. PLEMMONS, and A.H. SAMEH

Outline

1 Context

Meta-analysis covering:

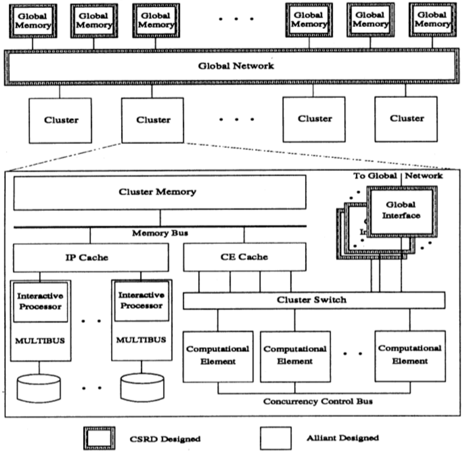
1. Parallel algorithms for dense matrix computations
2. Implementation practices
3. Efficiency analysis

2 Intro/Abstract

1. Efficient parallel algorithm design ought to be architecture-specific

2. Efficient algorithms can be decomposed into Computational Primitives

3 Architecture



Hierarchical shared memory and distributed memory architectures both influence algorithm design with a component denoted by Δ_l or the *data loading overhead*. So if we include *arithmetic time* T_a we get the following:

$$T = T_a + \Delta_l = n_a \tau_a + n_l \tau_l, \tag{1}$$

This is the basis of our analysis. Alternatively:

$$\frac{\Delta_l}{T_a} = \lambda \mu \tag{2}$$

where $\mu = n_l / n_a$ is the cache-miss ratio and $\lambda = \tau_l / \tau_a$ is the cost ratio.

4 Computational Primitives

BLAS

1. Basic Linear Algebra Subroutines (Subprograms)
2. Comprise the base computational units in LA

4.1 BLAS Level 1

Vector-Vector Operations

1. $\alpha \leftarrow x^T y$ (dot product)
2. $y \leftarrow y \pm \alpha x$ (vector triads)
3. Note : BLAS 1 requires many synchronizations relative to the number of arithmetic ops (large $\mu = n_l / n_a$).

4.2 BLAS Level 2

Matrix-vector Operations

1. $y \leftarrow y \pm Ax$ (matrix-vector product)
2. $A \leftarrow A \pm xy^T$ (rank-1 update)
3. BLAS 2 allows us to compute many BLAS 1 primitives in parallel thereby increasing n_a relative to n_l per process.
4. Note : BLAS 2 can degrade to BLAS1 as $dim(A)$ or $min(dim(A))$ goes to 1.

4.3 BLAS Level 3

Matrix-matrix Operations

1. $C \leftarrow C + AB$ (Matrix multiplication)
2. By Gallivan et al, typically the most efficient primitive IF cache size is considered when partitioning/decomposing the problem. Blocksize decision gives us maximum speed-up.

5 The Big Idea : Blocksize Analysis

Consider the BLAS3 primitive $C \leftarrow C + AB$. We would expect to partition the matricies $C, A,$ and B into submatricies C_{ij}, A_{ik} and B_{kj} whose dimensions are $m_1 \times m_3, m_1 \times m_2$ and $m_2 \times m_3$, respectively. Our basic loop might be of the form:

```
do i = 1, k_1
        do k = 1, k_2
                do j = 1, k_3
                    C_ij = C_ij + A_ik * B_kj
                end do
        end do
end do
```

where $n_1 = k_1 m_1, n_2 = k_2 m_2,$ and $n_3 = k_3 m_3$. Consider number of transers required for given submatricies:

$$\mu = \frac{1}{2m_1} + \frac{1}{2m_2} + \frac{1}{2n_3} \tag{3}$$

If infinite cache, we have a minimum of:

$$\mu = \frac{1}{2n_1} + \frac{1}{2n_2} + \frac{1}{2n_3} \tag{4}$$

We want to minimize m_1 and m_2 subject to number of processors and cache size....

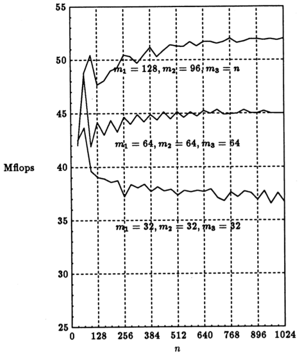
As it turns out this takes a form:

$$\mu = \frac{1}{\sqrt{CS}} + \frac{p}{2CS} + \frac{1}{2n_3}. \tag{5}$$

where CS is the cache size and assuming n_3 is larger than $\sqrt{CS}$.

5.1 Results

Performance for a square matrix multiplication on Alliant FX/8



6 Conclusion

1. Data Locality - The key factor in exploiting parallelism.

2. Blocksize - Main tool to control factors of Data Locality and ensure effective load management

Questions?