SIAL Course Lecture 5

Victor Lotrich, Mark Ponton, Erik Deumens,
Rod Bartlett, Beverly Sanders

AcesQC, LLC

QTP, University of Florida

Gainesville, Florida

Lecture 5: Algorithms

Balancing act

Three factors in parallel programming and software engineering

1. Data: Where is it and when?
2. Work: How much and who can do it?
3. Dependencies: Is data available when the worker is?

Case studies

- Consider three examples from electronic structure theory
  - They are representative and other domains see similar cases
  - Good performance, requires different approach in each case
  - Number of basis functions N
    - is a basic measure of size

Four cases

- "Standard" SIAL programming
- Poor performance because of poor balance
  1. SCF
  2. Perturbative (T)
  3. CCSD ring term

Data and Work

- SCF N^2
- MP2 N^4
  - data not stored
- CCSD N^4
- CCSD(T) N^6
  - data not stored in full

- SCF N^4
- MP2 N^5
  - energy only
- CCSD N^6
- CCSD(T) N^7
  - not iterative

SIAL data management

- served arrays: request/prepare
- distributed arrays: get/put
- Algorithms
  - Direct SCF & MP2: Small management
  - CCSD: Medium
  - Transformation, CCSD: Large
  - CCSD(T): Huge, if 6-index blocks are used

Typical data structures

- Rank 2 arrays (SCF)
  - p, q: AO or MO
  - A(p,q)
- Rank 4 arrays (CCSD, MP2, 2-el transf)
  - a, b, c: unoccupied/virtual MO
  - i, j, k, l: occupied MO
  - A(a,b,c,i), A(a,b,i,j), A(a,i,j,k), A(i,j,k,l)
- Rank 6 arrays: A(a,b,c,i,j,k)

Typical data blocks

- Segment size
  - AO: 30-40
  - Virtual MO: 30-40
  - Occupied MO: 28-30
    - typical molecules with for 80 or more electrons
- Block size
  - 2-index: < .1 MB
  - 4-index: 5-10 MB
  - 6-index: 1.7 – 13.8 GB **Too large!**

Standard case

“Standard SIAL” programming

- Example: CCSD
- Predominance of
  - rank 4 arrays
  - N^6 contractions
- Result
  - SIP hides communication behind computation
    - Small “block wait” time reported

Poor performance

- We consider three examples
  - Why they perform poorly
  - What to do about the performance
  - These show typical imbalances
  - And how to fix performance in each case

Poor balance case one

Poor balance: SCF

- Too little work: SCF
  - problem shows up on many processors
  - rank 2 arrays
  - too little work per block

“Standard SIAL” Fock build

- Integral computation and contraction are separate
- Each 8-fold symmetry contribution to 6 blocks of Fock matrix is separate
- Limited scaling beyond 4,000 cores

```
PARDO mu, nu, lambda, sigma
    GET D(lambda,sigma)
    compute_integrals AO(mu,nu,lambda,sigma)
    X(mu,vu) = AO(mu,nu,lambda,sigma) * D(lambda,sigma)
    PUT F(mu,nu) += X(mu,nu)
ENDPARDO mu, nu, lambda, sigma
```

Better balance Fock build

- Merge all operations on one integral block in one special super instruction
- Use STATIC arrays for holding D and building F
- Accumulate at the end
- Scales to much higher core count

```
PARDO mu, nu, lambda, sigma
    execute form_fock AO(mu,nu,lambda,sigma)
ENDPARDO mu, nu, lambda, sigma
# collect from all cores
```

Poor balance case two

Poor balance: (T)

- Too much data: perturbative triples "(T)"

$E = \Sigma_{abcijk}\ b^{abc}_{ijk}\ X^{abc}_{ijk}$

$X^{abc}_{ijk} = \Sigma_d\ t^{ad}_{ij}\ V^{bc}_{dk} - \Sigma_m\ t^{ab}_{im}\ V^{mc}_{jk}$

$b^{abc}_{ijk} = X^{abc}_{ijk}/(\varepsilon_a+\varepsilon_b+\varepsilon_c-\varepsilon_i-\varepsilon_j-\varepsilon_k)$

  - standard 6-index blocks are too large
  - leads to much waste on the block stack
  - too much communication floods the system

Two poor options

- Use simple indices for j and k of X^{abc}_{ijk}
  - Leads to a lot of data requests
- Use regular MOINDEX and smaller segments
  - Then the CCSD step in the calculation is inefficient, or
  - Expensive re-blocking data shuffle is needed

Third option

- SUBINDEX ii OF i
  - Blocks of t^{ad}_{ij} stay the same
  - LOCAL X(a,b,c,i,jj,kk)
    - X blocks have two small dimensions
    - And manageable size
  - Consistent with philosophy of blocking
  - Process in sub-blocks
  - Reduce communication compared to simple indices

SIAL implementation

- Big segments are processed
- Blocks are fetched to minimize communication
- Inner loops still have enough work

```
SUBINDEX ii OF i
SUBINDEX jj OF j
SUBINDEX kk OF k
PARDO a, b, c
    DO k
    DO j
        REQUEST Vaaai
        DO i
            REQUEST Taiai
            DO jj IN j
            DO ii IN i
                DO d
                . . .
                DO m
                . . .
ENDPARDO a, b, c
```

Another problem with (T)

- This code will run out of work
- PARDO a, b, c
- Has insufficient parallelism on 60,000 cores
- SIP has “fall through” load balancing

SIP load balancing

- Work is divided among tasks
- Distribute index-sets to workers
- When a worker is done, it asks more
- If no more, it goes to the next stmt
- If not a barrier, then worker works more!

```
PARDO a, b, c
      # work 1
      . . .
ENDPARDO a, b, c
PARDO a, b, c
      # work 2
      . . .
ENDPARDO a, b, c
PARDO a, b, c
      # work 3
      . . .
ENDPARDO a, b, c
. . .
```

Careful orchestration

- Adding k and j to PARDO is no good
  - Too much communication
- Split the work in the i, j, k inner loops
  - Make multiple PARDO a, b, c
  - Now 60,000 cores are grouped per PARDO
  - Each group working in a PARDO on a different part of the inner loop
- Next generation SIP will do this by itself

Poor balance case three

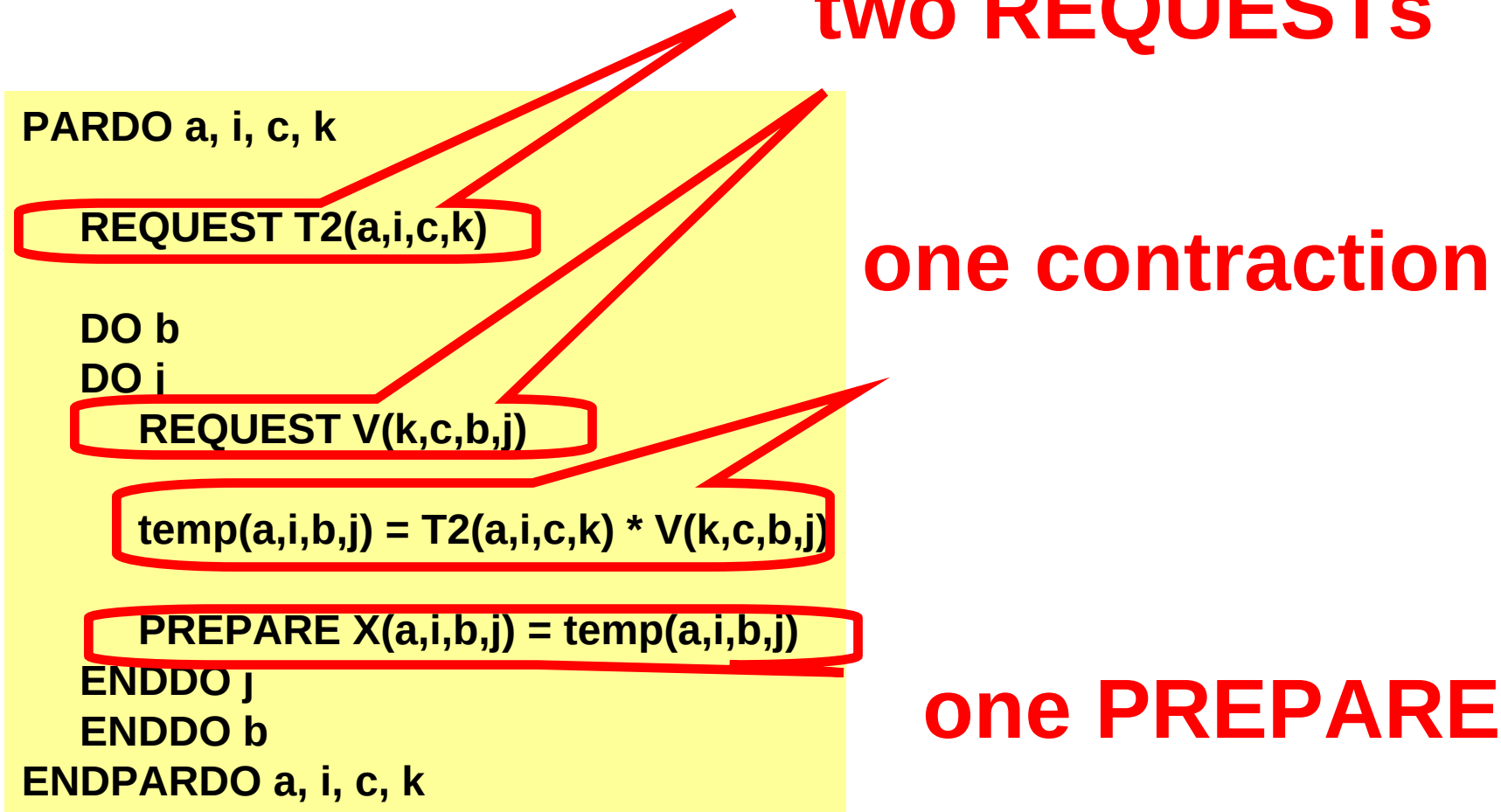
Poor balance: ring

- Too many communications: CCSD “ring”

$X^{ab}_{ij} = \Sigma_{ck}\ t^{ac}_{ik}\ V^{kb}_{cj}$

  - too many requests for data
  - too little work
  - usually no problem, because evaluation is quick
  - block-wait time is large fraction of total time

“Standard SIAL” ring contraction



```
PARDO c, k
    ALLOCATE L1(*,*,c,k)
    DO a
    DO i
        REQUEST T2(a,i,c,k)
        L1(a,i,c,k) = T2(a,i,c,k)
    ENDDO i
    ENDDO a
    DO b
    DO j
        REQUEST V(k,c,b,j)
        DO a
        DO i
            temp(a,i,b,j) = L1(a,i,c,k) * V(k,c,b,j)
            PREPARE X(a,i,b,j) = temp(a,i,b,j)
        ENDDO i
        ENDDO a
    ENDDO j
    ENDDO b
    DEALLOCATE L1(*,*,c,k)
ENDPARDO c, k
```

cache a local copy

use in b, j loop

less parallel, still faster

Take-home message

- There are more ways than one to handle a problem
- Different algorithms are needed for different ranges of
  - problem size
    - amount of data
    - amount of work
  - number of cores