Super instruction architecture of a parallel implementation of coupled cluster theory

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

AcesQC, LLC

QTP, University of Florida

Gainesville, Florida

Talk outline

- Design of petascale capable software
  - What is Super Instruction Architecture
  - How do SIAL and SIP work?
    - like Java and JavaVM
- SIAL programming
  - What do SIAL programs look like?
  - SIAL programs are efficient
  - SIAL programming is productive

ACES III software

- Developed since 2003
- Parallel for shared and distributed memory
- Capabilities
  - Hartree-Fock (RHF, UHF)
  - MBPT(2) energy, gradient, hessian
  - CCSD(T) energy and gradient (DROPMO)
  - EOM-CC excited state energies
- Details in talk by Prof. Bartlett this afternoon

Serial computers

- CPU (core) is in control
  - Performs operations on 64 bit quantities
  - Uses 64 bit registers
- Coordinates data movement
  - RAM
  - DASD
  - Network

Parallel computers

- Multiple CPUs (cores) must coordinate
- Data management is more complex
  - Data sharing
  - Data locks
  - Data flow delays

Parallel programming

- Some design patterns
  - Master-worker
    - Master tells workers what to do
  - Client-server
    - Clients ask one or more servers what to do
- Data sharing
  - Between two or few CPUs
    - exchange messages or share a token or lock
  - Between all or most CPUs
    - Post barriers and send broadcast messages

SIA = super serial

- Super Instruction Architecture
- First guiding principle
  - **Parallel computer = “super serial” computer**
    - Number <-> super number = block
      - 64 bit <-> 640,000 bit
    - CPU operation -> super instruction = subroutine
    - Data movement to RAM, DASD, network -> move blocks to local or remote locations

SIA = simple code

- Second guiding principle
  - **Separate algorithm from execution**
    - Define a simple language to express the algorithm
    - Leave details of execution to a lower level
  - The first principle allows just that
    - Because every block operation takes finite time
    - No operation can be considered instantaneous
    - All operations count in scheduling

A computer with a single CPU

- Basic data item: 64 bit number
- High level language: Fortran, C
  - c = a + b
- Assembly language
  - ADD dest,src
  - ADD is an operation code
  - dest and src are registers

A parallel computer running SIAL

- Super Instruction Assembly Language **SIAL**
  - R(I,J,K,L) += V(I,J,C,D) * T(C,D,K,L)
- **Bytecode**
- Super Instruction Processor **SIP**
  - Fortran/C/MPI code
- Hardware execution
  - x86_64 PowerPC

- **Java**
  - R(I,j,k,l) += V(I,j,c,d) * T(c,d,k,l);
- **Bytecode**
- **JavaVM**
  - C code
- Hardware execution
  - x86_64, PowerPC

A parallel computer running SIAL

- Super Instruction Assembly Language **SIAL**
  - R(I,J,K,L) += V(I,J,C,D) * T(C,D,K,L)
- Bytecode
- Super Instruction Processor SIP
  - Fortran/C/MPI code
- Hardware execution

- **Java**

program

  - R(I,j,k,l) += V(I,j,c,d) * T(c,d,k,l);
- Bytecode
- JavaVM
  - C code
- Hardware execution

A parallel computer running SIAL

- Super Instruction Assembly Language SIAL
  - R(I,J,K,L) += V(I,J,C,D) * T(C,D,K,L)
- **Bytecode**
- Super Instruction Processor SIP
  - Fortran/C/MPI code
- Hardware execution

compile

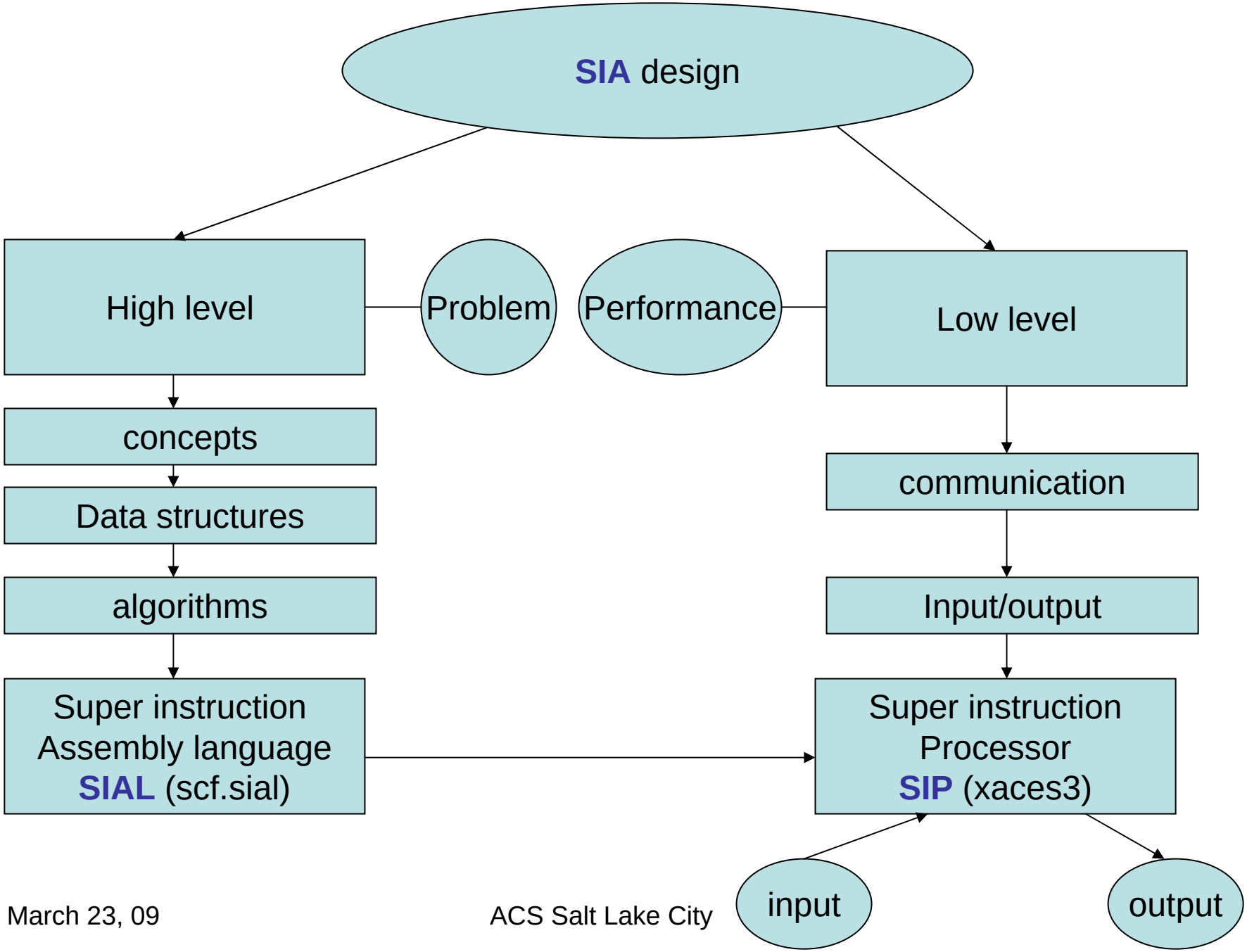
- Java
  - R(I,j,k,l) += V(I,j,c,d) * T(c,d,k,l);
- **Bytecode**
- JavaVM
  - C code
- Hardware execution

A parallel computer running SIAL

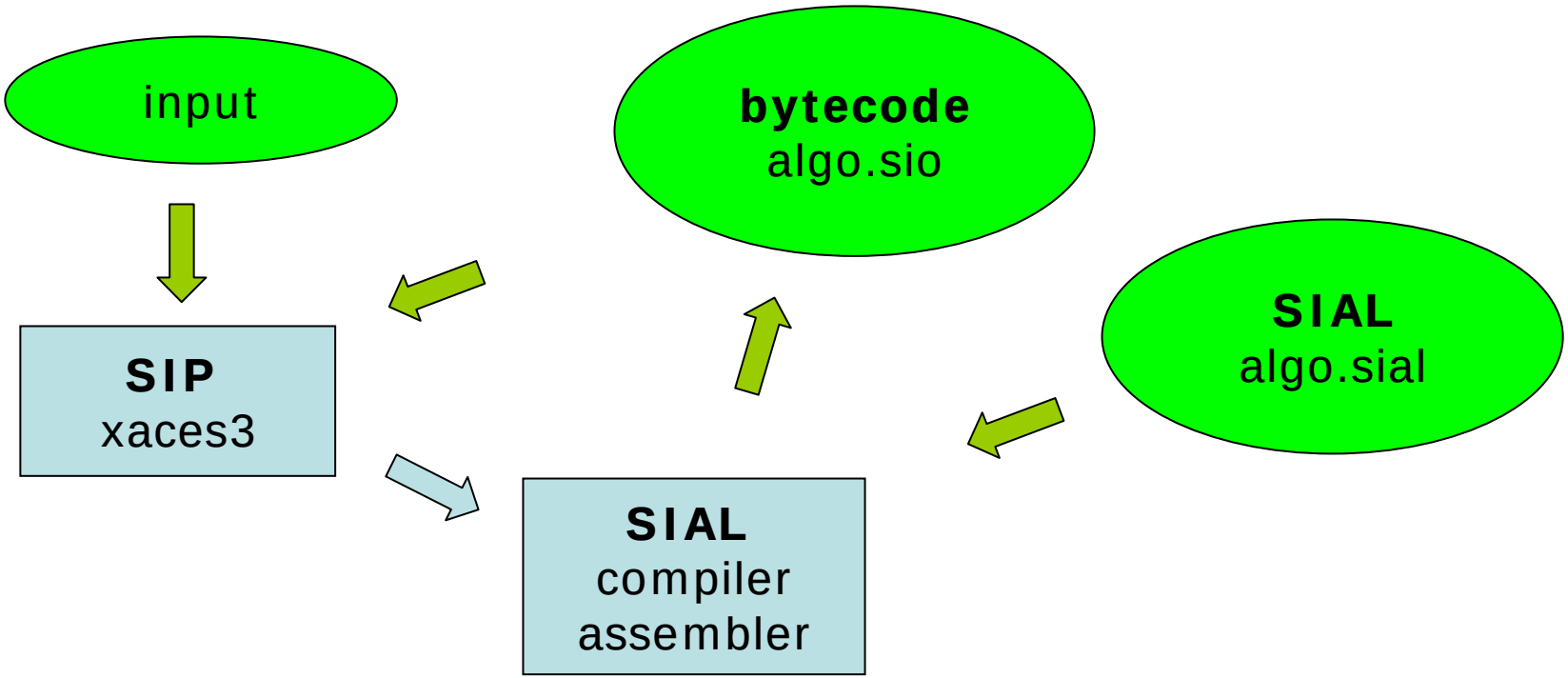
- Super Instruction Assembly Language SIAL
  - R(I,J,K,L) += V(I,J,C,D) * T(C,D,K,L)
- Bytecode
- Super Instruction Processor **SIP**
  - Fortran/C/MPI code
- Hardware execution

execute

- Java
  - R(I,j,k,l) += V(I,j,c,d) * T(c,d,k,l);
- Bytecode
- **JavaVM**
  - C code
- Hardware execution


SIA design
High level
Problem
Performance
Low level
concepts
Data structures
algorithms
Super instruction
Assembly language
SIAL (scf.sial)
communication
Input/output
Super instruction
Processor
SIP (xaces3)
input
output

User level execution flow



Coarse grain parallelism

- While executing super instructions in SIAL program
- Example: memory super instruction
  - GET block
  - Can be from
    - Local node RAM
    - Other node RAM
  - Time for data to become available differs

Fine grain parallelism

- While executing individual super instructions
- Example: contractions and integrals
  - * (contractions)
  - compute_integrals
  - Can use multiple cores
  - Can use accelerators
    - GPGPUs and Cell processors
    - FPGAs (field programmable gate arrays)

Super instruction flow

Worker i

- GET a -> ask j
- …
- d=b*c
- … wait for a?
- a arrives <-
- e=a*d
- …

Worker j

- …
- <- send a
- …
- …
- …
- …
- …

Distributed data

- N worker tasks
  - each with local RAM
- Data distributed in RAM of workers
  - AO-based: direct use of integrals
  - MO-based: use transformed integrals
- Array blocks are spread over all workers

Served (disk resident) data

- N worker tasks
  - each with local RAM
- M server tasks
  - have access to local or global disk storage
  - accept, store and retrieve blocks
- Data served to and from disk
  - workers only have indirect access to disk
  - through servers

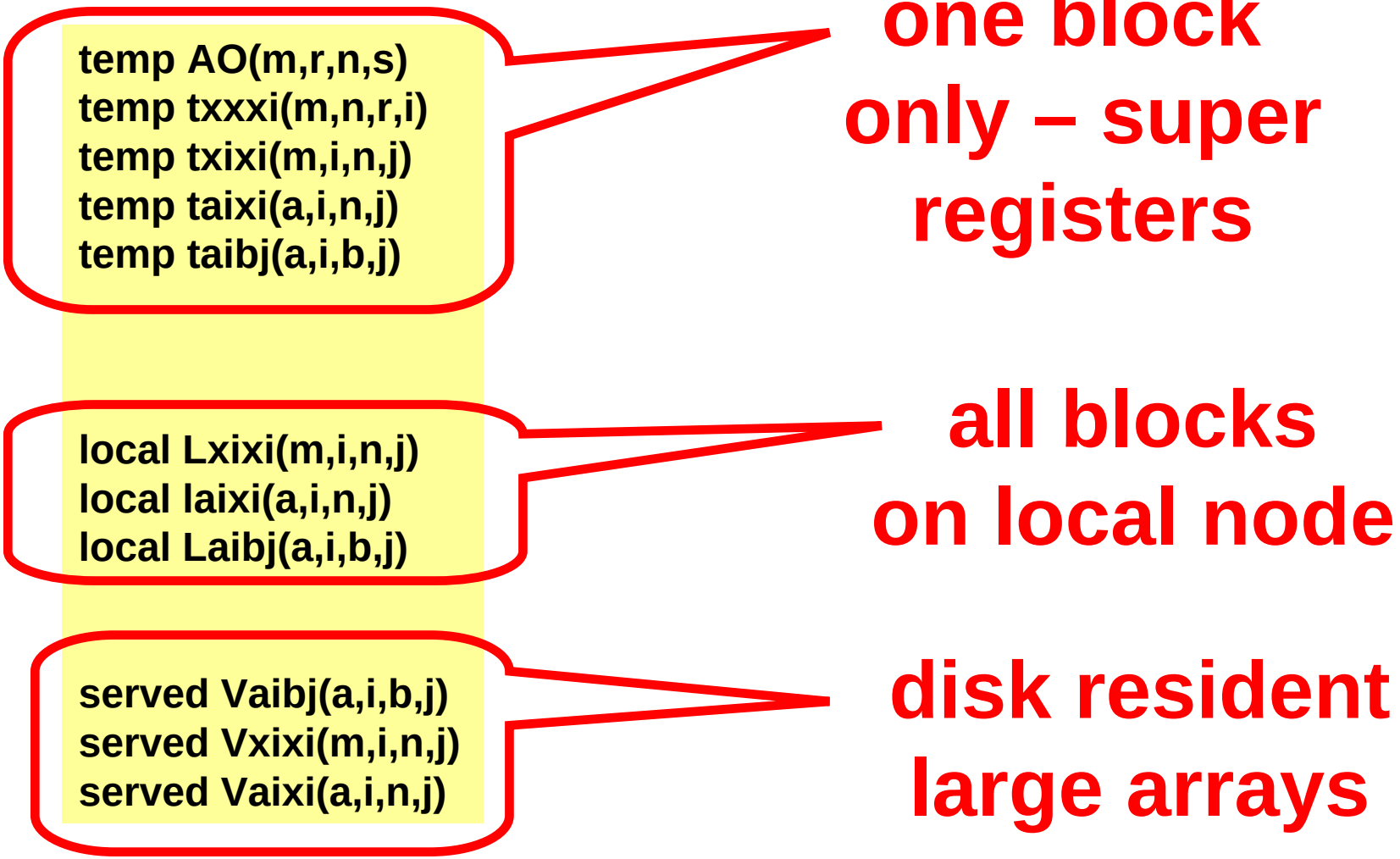
A SIAL program

```
SIAL 2EL_TRANS

aoindex m = 1, norb
aoindex n = 1, norb
aoindex r = 1, norb
aoindex s = 1, norb

moaindex a = baocc, eaocc
moaindex b = baocc, eaocc
moaindex i = bavirt, eavirt
moaindex j = bavirt, eavirt
```

program declaration

declaration of block indices

Pulay algorithm for two-electron integral transformation.


temp AO(m,r,n,s)
temp txxxi(m,n,r,i)
temp txixi(m,i,n,j)
temp taixi(a,i,n,j)
temp taibj(a,i,b,j)
one block only – super registers
local Lxixi(m,i,n,j)
local laixi(a,i,n,j)
local Laibj(a,i,b,j)
all blocks on local node
served Vaibj(a,i,b,j)
served Vxixi(m,i,n,j)
served Vaixi(a,i,n,j)
disk resident large arrays

```
PARDO m, n
        allocate Lxixi(m,*,n,*)
        DO r
        DO s
            compute_integrals AO(m,r,n,s)
            DO j
                txxxi(m,r,n,j) = AO(m,r,n,s)*c(s,j)
            DO i
                txixi(m,i,n,j)  = txxxi(m,r,n,j)*c(r,i)
                Lxixi(m,i,n,j) += txixi(m,i,n,j)
             ENDDO i
             ENDDO j
         ENDDO s
         ENDDO r
         DO i
         DO j
             PREPARE Vxixi(m,i,n,j) = Lxixi(m,i,n,j)
         ENDDO j
         ENDDO i
         deallocate Lxixi(m,*,n,*)
ENDPARDO m, n
execute server_barrier
```

```
PARDO m, n
        allocate Lxixi(m,*,n,*)
        DO r
        DO s
            compute_integrals AO(m,r,n,s)
            DO j
                txxxi(m,r,n,j) = AO(m,r,n,s)*c(s,j)
            DO i
                txixi(m,i,n,j)  = txxxi(m,r,n,j)*c(r,i)
                Lxixi(m,i,n,j) += txixi(m,i,n,j)
             ENDDO i
             ENDDO j
         ENDDO s
         ENDDO r
         DO i
         DO j
             PREPARE Vxixi(m,i,n,j) = Lxixi(m,i,n,j)
         ENDDO j
         ENDDO i
         deallocate Lxixi(m,*,n,*)
ENDPARDO m, n
execute server_barrier
```

parallel over block indices m and n

```
PARDO m, n
        allocate Lxixi(m,*,n,*)
        DO r
        DO s
            compute_integrals AO(m,r,n,s)
            DO j
                txxxi(m,r,n,j) = AO(m,r,n,s)*c(s,j)
            DO i
                txixi(m,i,n,j)  = txxxi(m,r,n,j)*c(r,i)
                Lxixi(m,i,n,j) += txixi(m,i,n,j)
             ENDDO i
             ENDDO j
         ENDDO s
         ENDDO r
         DO i
         DO j
             PREPARE Vxixi(m,i,n,j) = Lxixi(m,i,n,j)
         ENDDO j
         ENDDO i
         deallocate Lxixi(m,*,n,*)
ENDPARDO m, n
execute server_barrier
```

allocate partial local array

delete local array

```
PARDO m, n
        allocate Lxixi(m,*,n,*)
        DO r
        DO s
          compute_integrals AO(m,r,n,s)
          DO j
             txxxi(m,r,n,j) = AO(m,r,n,s)*c(s,j)
          DO i
             txixi(m,i,n,j)  = txxxi(m,r,n,j)*c(r,i)
             Lxixi(m,i,n,j) += txixi(m,i,n,j)
           ENDDO i
           ENDDO j
        ENDDO s
        ENDDO r
        DO i
        DO j
           PREPARE Vxixi(m,i,n,j) = Lxixi(m,i,n,j)
        ENDDO j
        ENDDO i
        deallocate Lxixi(m,*,n,*)
ENDPARDO m, n
execute server_barrier
```

compute integral block

```
PARDO m, n
        allocate Lxixi(m,*,n,*)
        DO r
        DO s
            compute_integrals AO(m,r,n,s)
            DO j
                txxxi(m,r,n,j) = AO(m,r,n,s)*c(s,j)
            DO i
                txixi(m,i,n,j)  = txxxi(m,r,n,j)*c(r,i)
                Lxixi(m,i,n,j) += txixi(m,i,n,j)
             ENDDO i
             ENDDO j
        ENDDO s
        ENDDO r
        DO i
        DO j
            PREPARE Vxixi(m,i,n,j) = Lxixi(m,i,n,j)
        ENDDO j
        ENDDO i
        deallocate Lxixi(m,*,n,*)
ENDPARDO m, n
execute server_barrier
```

transform two indices into local array using same integrals

```
PARDO m, n
        allocate Lxixi(m,*,n,*)
        DO r
        DO s
            compute_integrals AO(m,r,n,s)
            DO j
                txxxi(m,r,n,j) = AO(m,r,n,s)*c(s,j)
            DO i
                txixi(m,i,n,j)  = txxxi(m,r,n,j)*c(r,i)
                Lxixi(m,i,n,j) += txixi(m,i,n,j)
             ENDDO i
             ENDDO j
         ENDDO s
         ENDDO r
         DO i
         DO j
           PREPARE Vxixi(m,i,n,j) = Lxixi(m,i,n,j)
         ENDDO j
         ENDDO i
         deallocate Lxixi(m,*,n,*)
ENDPARDO m, n
execute server_barrier
```

store in served array

```
PARDO m, n
        allocate Lxixi(m,*,n,*)
        DO r
        DO s
            compute_integrals AO(m,r,n,s)
            DO j
                txxxi(m,r,n,j) = AO(m,r,n,s)*c(s,j)
            DO i
                txixi(m,i,n,j)  = txxxi(m,r,n,j)*c(r,i)
                Lxixi(m,i,n,j) += txixi(m,i,n,j)
             ENDDO i
             ENDDO j
         ENDDO s
         ENDDO r
         DO i
         DO j
             PREPARE Vxixi(m,i,n,j) = Lxixi(m,i,n,j)
         ENDDO j
         ENDDO i
         deallocate Lxixi(m,*,n,*)
ENDPARDO m, n
execute server_barrier
```

wait for all workers to finish storing

```
PARDO n, i, j
        allocate Laixi(*,i,n,j)
        DO m
            REQUEST Vxixi(m,i,n,j) m
        DO a
            taixi(a,i,n,j) = Vxixi(m,i,n,j)*c(m,a)
            Laixi(a,i,n,j) += taixi(a,i,n,j)
        ENDDO a
        ENDDO m

        DO a
            PREPARE Vaixi(a,i,n,j) = Laixi(a,i,n,j)
        ENDDO a
        deallocate Laixi(*,i,n,j)
ENDPARDO n, i, j
execute server_barrier
```

retrieve block from servers/disk

transform third index

```
PARDO a, i, j
        allocate Laibj(a,i,*,j)
        DO n
            REQUEST Vaixi(a,i,n,j) n
        DO b
            taibj(a,i,b,j) = Vaixi(a,i,n,j)*c(n,b)
            Laibj(a,i,b,j) += taibj(a,i,b,j)
         ENDDO b
         ENDDO n

         DO b
             PREPARE Vaibj(a,i,b,j) = Laibj(a,i,b,j)
         ENDDO b
         deallocate Laibj(a,i,*,j)
ENDPARDO a, i, j
execute server_barrier

ENDSIAL 2EL_TRANS
```

transform fourth imdex

store final integrals in served array

SIAL performance

- All super instructions are asynchronous
- Thus execution is very elastic
- Helps maintain consistent performance on many parallel architectures

SIP optimization and tuning

- Optimize with traditional techniques
  - optimize the basic contraction operations by mapping them to DGEMM calls
  - create fast code to generate integrals
  - optimize memory allocation by using multiple block stacks
  - optimize execution and data movement

Performance comparisons

- Invitation to compare performance on some problems
  - http://www.qtp.ufl.edu/PCCworkshop/PCCbenchmarks.html
- Not the fastest on small number of cores
- Very reliable
  - ACES III works when others cannot run because of design limitations
- Very predictable performance and scaling

SIAL programmer productivity

- SIAL has simple syntax
  - Experience shows it is very expressive
- Exact data layout is done by SIP
  - Allows runtime tuning and optimization
- SIAL has rich set of data structures
  - temporary, local, distributed, and served arrays
- SIAL specific IDE in development
  - integrated in Eclipse

Productivity comparisons

- Other tools for parallel development
  - UPC (Universal Parallel C)
  - CAF (Co-Array Fortran)
  - GA (Global Array Tools)
  - DDI (Distributed Data Interface)
- Simple syntax
- Specify precise data layout
  - PGAS partitioned global address space
  - Rigorous array blocking

Towards petascale computing

- ACES III
  - Ready for real work
  - Has run on 8,192 processors
- SIAL
  - Useful in electronic structure
  - Approach can be used in other domains
- Downloads and more at
  - http://www.qtp.ufl.edu/ACES