SIAL Course Lectures 1 & 2

Victor Lotrich, Mark Ponton, Erik Deumens, Rod Bartlett, Beverly Sanders
QTP University of Florida
AcesQC, LLC
Gainesville Florida

Training goals

- Understand the parallel architecture behind the language
- Learn SIAL language elements
- Learn to program in SIAL
- Apply the programming strategy
  - Decide on the algorithm
  - Analyze the algorithm
  - Design the implementation
  - Write the SIAL program
  - Write a test program

Course overview

- Lectures
  - Super instruction architecture (SIA) for software design
  - Language definition for SIAL
  - The workings of SIP
  - Performance
  - Algorithms

Course overview

- Workshop
  - Study a selection of SIAL programs that show the capabilities
  - Modify them
  - Run them

Lecture 1: Architecture

Serial computers: processing

- CPU or core
    - Central processing unit
  - Is always in control
  - Executes instructions
  - Performs
    - Logical tests
    - Integer and float operations

Serial computers: data

- Moves data to and from RAM
    - Random access memory
- Directs data to and from DASD
    - Direct access storage device
- Sends data to and from networks
    - Communication with remote processors and storage

Parallel computers: processing

- Multiple CPUs or cores
  - Must coordinate activity
  - All processing requires communication
  - Some general types of processing
    - Master – worker
      - Master tells workers what to do
    - Client - server
      - Clients ask one or more servers what to do

Parallel computers: data

- All cooperating processors need data
- Managing the data requires coordination
  - Naturally parallel
    - Every computation works on a pre-assigned piece
  - Work on shared data
    - Requires synchronization primitives
      - Between two or few CPUs
        - exchange messages or share a token or lock
      - Between all or most CPUs
        - Post barriers and send broadcast messages

Programming

- Serial programming
  - Compose for piano, guitar, or flute
- Parallel programming
  - Compose for quartet or ensemble
    - Multi-core parallelism: 4 to 8 way
  - Compose for a philharmonic orchestra
    - Massive parallelism: 1,000 to 100,000 way

Modern computers

- Problem with modern computers
  - CPUs (cores) are **very** fast (x 1,000)
  - RAM is **much** slower (x 1)
  - DASD and network are **glacial** in comparison (x .001)
- Need for delicate orchestration of
  - Processing of data
  - Movement of data

SIA: Super Serial

- Super Instruction Architecture
- First guiding principle
  - Parallel computer = “super serial” computer
    - Number -> super number = block
    - CPU Operation -> super instruction = subroutine
    - Data movement to RAM, DASD -> move blocks to local or remote locations

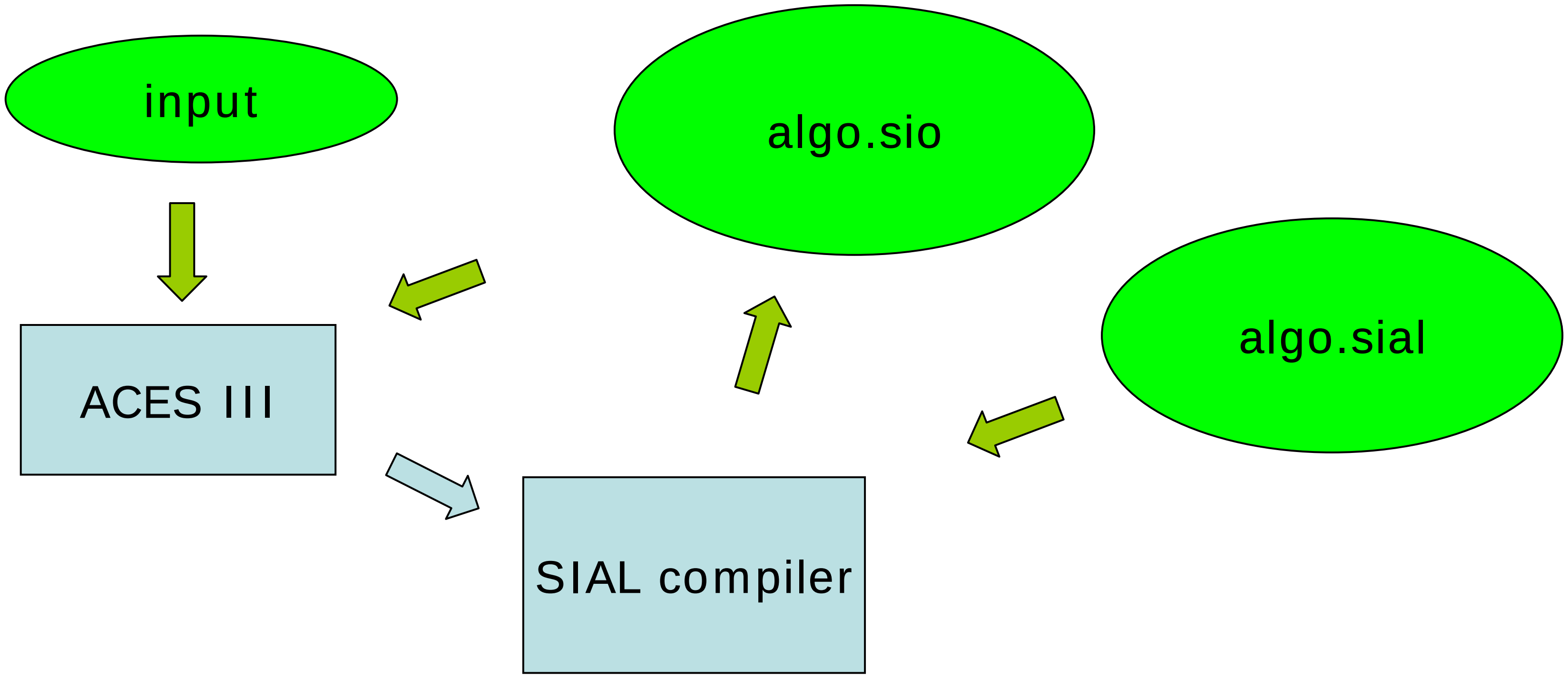
SIA: simple code

- 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 time
      - No operation is significantly faster than any other operation
      - (As is the case in a modern microprocessor.)

SIA: How does it work?

- Analyze the problem
  - Pick the algorithms
- Write SIAL program
    - Super instruction assembly language
  - Details of parallelism are not visible
- SIP runs the SIAL program
    - Super instruction processor
  - Knows and optimizes use of hardware
  - User can tune execution

Execution flow



Lecture 2: Language

SIAL Programmer Guide

- The full manual is available online
  - http://www.qtp.ufl.edu/ACES
  - -> Documentation
- SIAL is case insensitive like Fortran, not case sensitive like C or Java
- Lines are 256 characters and cannot be continued
- # marks that the rest is a comment

Constants

- Universal constants
  - Defined from ZMAT or JOBARC
    - scfenerg <- JOBARC
    - scfiter <- SCF_MAXCYC in ZMAT
- MO constants
  - These count in segments, not orbitals
    - nocc, naocc, nbocc
  - nocc and naocc are of different type

More constants

- Static c(mu,p)
  - No-spin orbital transformation matrix
  - Mu is MOINDEX 1:norb
  - P is AOINDEX 1:norb
  - Written by SCF program to JOBARC
  - Read from JOBARC by other programs

Predefined special instructions

- There is a list of special instructions
  - Block_to_list X
    - Write the blocks of array X to a file for reading by later programs, or the same one for restart.

Declarations

- MOINDEX p=1,nocc
  - Declares p as an index counting segments in the MO orbital space without spin
  - MOINDEX q=1,naocc is an error because naocc is for alpha spin
  - The actual range of segments is unknown until run time.
- INDEX i=1,20
  - Declares a simple index, e.g. for loop control

Declarations

- SCALAR a
  - A single floating point number
- STATIC a(p,q)
  - An array allocated locally for each core with a separate malloc intended to be static
  - p and q must have been declared first

Declarations

- LOCAL b(p,q)
  - An array allocated locally on the block stack so that it can be created and deleted
  - Are explicitly ALLOCATEd and DEALLOCATEd to manage RAM usage (local operations)
  - Can be partially ALLOCATEd to save space ALLOCATE b(*,p)

Declarations

- TEMP c(a,b,d,e)
  - A single block of data that is stored locally and can be used as if it is an array in multi-step computations
  - The block behaves like a super register
  - There is no coherence between the same TEMP on different processors

Declarations

- DISTRIBUTED v(mu,nu,p,q)
  - All blocks are distributed among all cores
  - The distribution is deterministic but unknown to the SIAL programmer, it is determined at run time
  - Accessed with PUT and GET
  - Are explicitly CREATEd and DELETEd to manage RAM usage (global operations)

Declarations

- SERVED s(mu,nu,lambda,sigma)
  - All blocks are written to DASD
  - The DASD are managed by a group of dedicated cores who manage moving the blocks between their RAM and DASD and send them to worker cores
  - Accessed with PREPARE and REQUEST
  - Implicitly created with PREPARE and explicitly DESTROYed

Control statements

- SIAL myprog / ENDSIAL myprog
  - Main SIAL program
- PROC suba / ENDPROC suba
  - SIAL procedure
  - Procedures must be declared inside a program
  - Organize code
  - Execute in global scope
- RETURN
- CALL suba

Control statements

- PARDO mu,nu,lambda,sigma WHERE MU<NU
  - Distribute the work inside the body over available worker cores
  - Assigning sets of index values to each core
  - Load balancing algorithms are used to optimize performance at run time

Control statements

- DO j / END DO j
  - Simple iteration over all values of I
- CYCLE j
  - Skip the rest of the DO-body and start at the top with the next value of j
- EXIT j
  - Exit the DO / ENDDO block and resume execution after the block

Control statements

- IF test / ELSE / ENDIF
  - Conditional execution of code blocks
  - Conditions are of the form k < j, etc.

Operations

- + - * ^ == < > <= >= && | !
- += -= *=
- ALLOCATE v(mu,*,nu,*)
  - Allocate the blocks of a LOCAL array
- DEALLOCATE v

Operations

- CREATE w / DELETE w
  - Allocate and free all blocks of a distributed array w
- PUT w(mu,nu,p,q) += tmp1(mu,nu,p,q)
  - Add block of tmp1 to block of w
- GET w(mu,nu,p,q)
  - Initiate transfer of a block of w

Operations

- PREPARE s(mu,nu,p,q) = tmp1(mu,nu,p,q)
  - Store block of tmp1 in served array s
- REQUEST s(mu,nu,p,q) nu
  - Initiate transfer of a block of s
  - Argument nu is “fast” index
  - Allows SIP to pre-fetch
- DESTROY s

Operation statements

- v3(p,q,r,s) = v2(p,q,r,mu) * c(mu,s)
    - Is an assignment and a contraction
- v3(p,q,r,s) = v1(p,q) * v2(r,s)
    - Is an assignment and a tensor product
- COLLECTIVE a+=b
    - All cores add b to a

Operation statements

- EXECUTE specinstr arg1,arg2,arg2
  - Execute the named special instruction
  - Supply arguments
  - The instruction must be registered and compiled
  - The arguments must match what the instructions expects
  - No checking is done, the instruction must do its own checking if necessary

Programming hints

- SERVED arrays should be large
  - If they are small, they may reside in RAM of the workers
  - Then they behave like DISTRIBUTED arrays, which are preferred
- Avoid accessing arrays inside inner loops
  - Staging blocks can have good performance impact
    - from DISTRIBUTED into TEMP
    - from SERVED into DISTRIBUTED into TEMP

Programming hints

- Accessing individual numbers inside blocks
  - Is like “super bit manipulation” slow
  - Requires special instruction
    - Example: energy_denominator
  - Should be rare to maintain good performance

Programming hints

- PARDO lambda,mu
  - Load balancing works well, but is not perfect
  - Think carefully about putting IF inside
    - A restriction lambda or mu inside a PARDO
    - Results in now work for some cores