ACES 3: Efficient Parallel Implementation of MBPT(2) and CCSD Energy, Gradient and Hessian Calculations

Erik Deumens

AcesQC and University of Florida

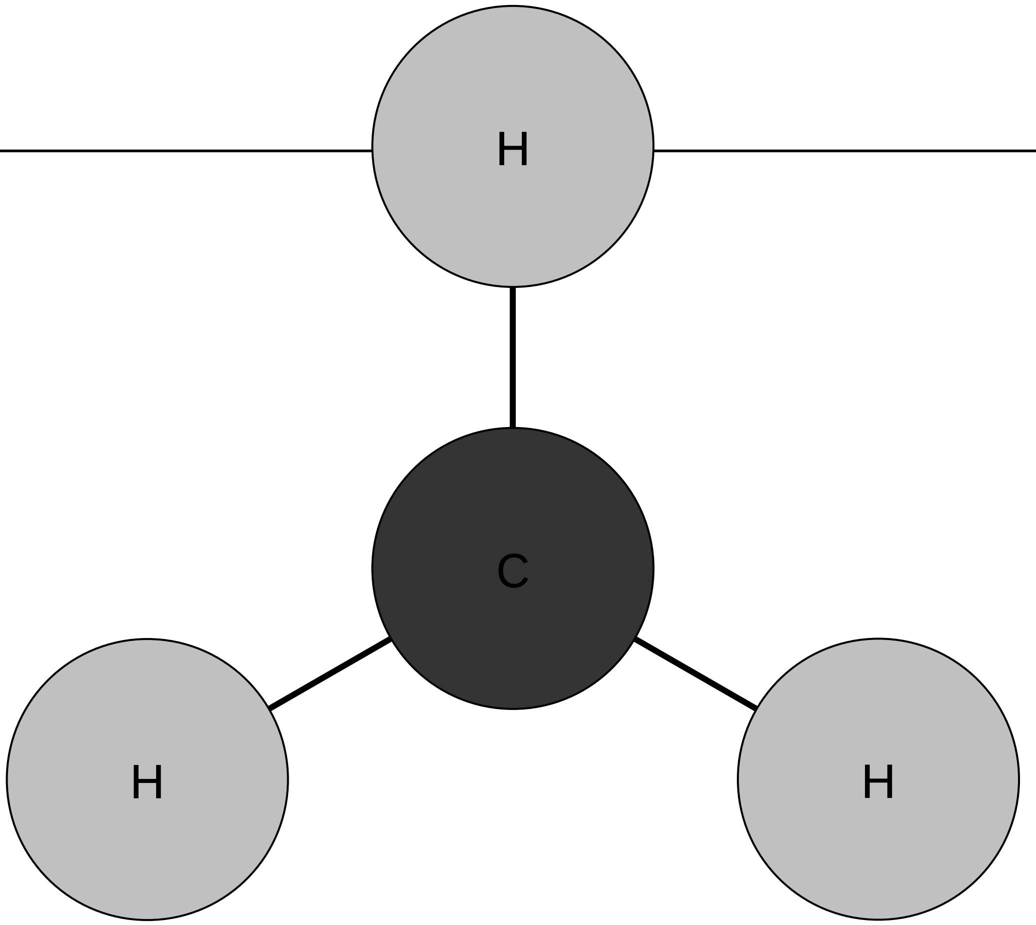
Jun 29, 2006

Outline of the talk

- What does ACES 3 do?
  - Computational chemistry
- How does it work in parallel?
  - Computer science and engineering
- Some examples
  - Performance analysis

What does ACES 3 do?

- Computational chemistry
  - Dynamics and structure of molecules
  - Atomic nuclei move slower
  - Electrons are fast like mosquitoes buzzing around a hiker


H
C
H
H

Computational chemistry

- Potential Energy Surfaces
  - Minima for stable molecular states
  - Saddle points for transition states
  - Reaction paths
- Need to compute
  - Energy
  - Gradient
  - Hessian

Serial to parallel

- ACES 2
  - Serial code
  - Developed since 1990
- ACES 3
  - Developed under CHSSI CBD-03
  - Parallel code for compute intense components
    - MBPT(2) energy, gradient, hessian
    - CCSD energy and gradient

Outline of the talk

- What does ACES 3 do?
  - Computational chemistry
- **How does it work in parallel?**
  - **Computer science and engineering**
- Some examples
  - Performance analysis

Why is this problem hard?

- CCSD calculations are compute and data intensive

  Large number of T amplitudes

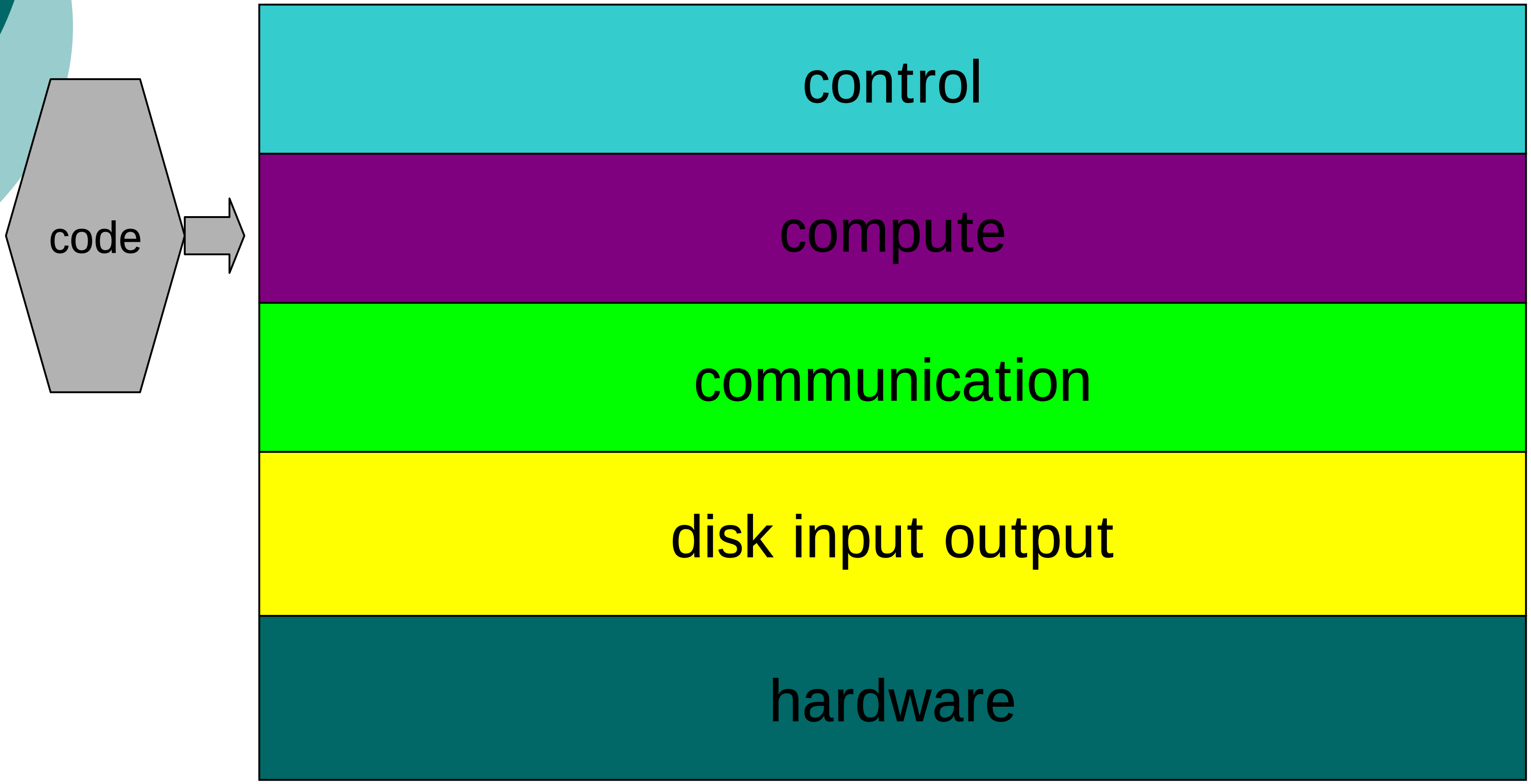
  Large numbers of integrals

  - to be kept in RAM, or on disk: stored method
  - to be computed multiple times: direct method

Computer Science and Engineering

- Need sophisticated design
  - Exploit parallelism
  - Feasible to write and debug
  - Possible to tune on multiple architectures
    - Distributed memory
      - Ratio of CPU speed vs. communication speed
    - Shared or NUMA memory
  - Easy to maintain

Traditional Design



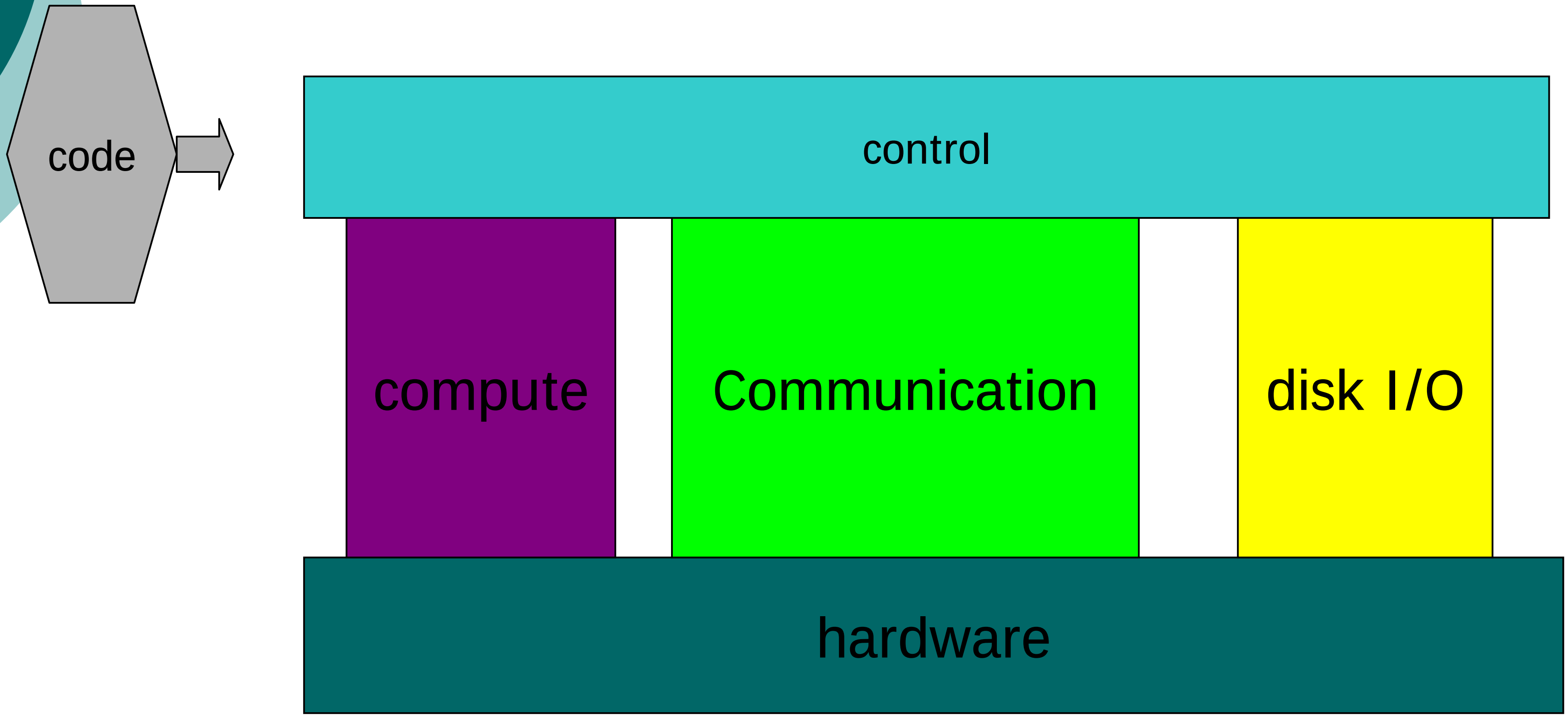
Traditional Design

- Assumptions
  - Data access latency and bandwidth
  - Computation intertwined with communication
  - Size for data that can be replicated
  - Hardware characteristics must fall in certain ranges to reach performance goals

Traditional Design

- Consequences
  - Detailed analysis by programmer
  - Match data flow with work flow
  - Manage communication deep in code

ACES 3 Design



ACES 3 Design

- Requirement
  - Allow flexibility to control separately at run-time:
    1. Computation
    2. Communication
    3. Disk input and output

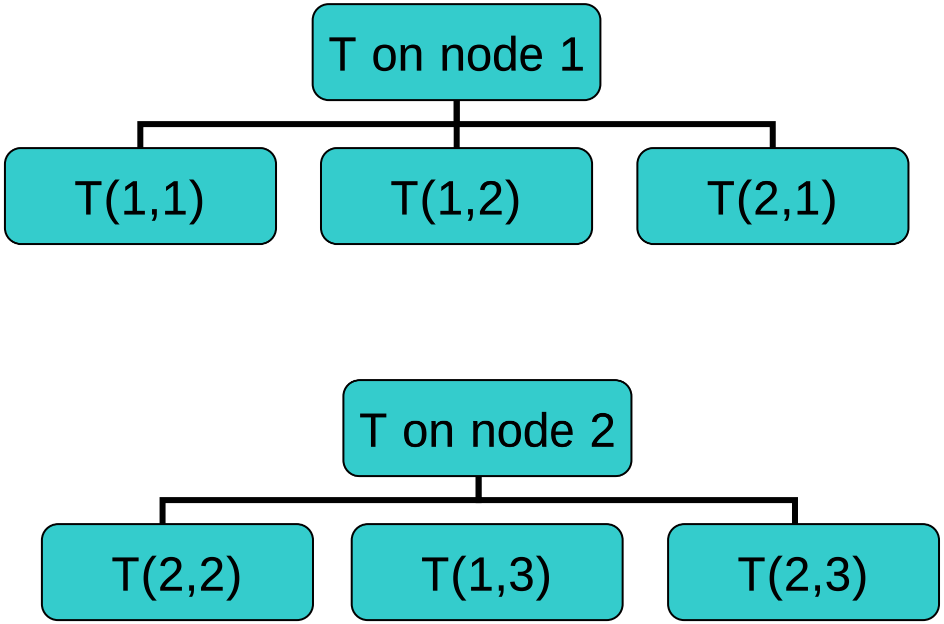
ACES 3 Design

- Principles
  - Define units of data
    - For movement and computation
  - Define basic operations on data units
    - All movement is asynchronous
  - Schedule operations and movement
    - Optimize hiding communication behind computation for every machine
    - Optimize data size to make its computation longer than its transportation

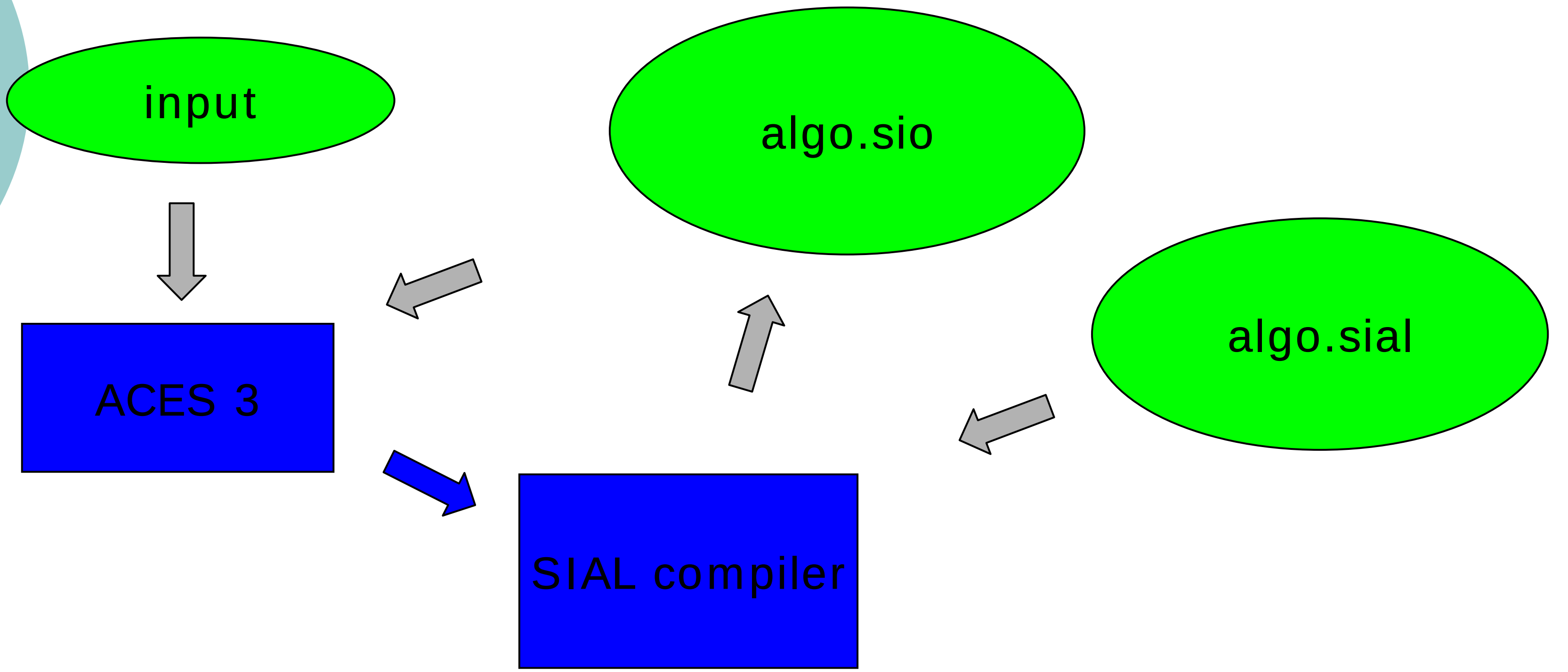
Data organization: numbers

T_{11}	T_{12}	T_{13}	T_{14}
T_{21}	T_{22}	T_{23}	T_{24}
T_{31}	T_{32}	T_{33}	T_{34}
T_{41}	T_{42}	T_{43}	T_{44}

Data organization: blocks



ACES 3 execution



Parallel architecture

- Distributed data in RAM of workers
  - AO direct use of integrals
  - MO use transformed integrals
- N worker tasks each with 1 GB RAM
- Array blocks are spread over all workers
- Workers compute integrals when integral instruction is called

Parallel architecture

- Served data to and from disk
  - AO no transformation of integrals
  - MO use transformed integrals
- N worker tasks and M server tasks
  - Workers are as before
  - Servers have disk cache and disk
  - Servers take and give blocks
  - Servers compute integrals when asked

ACES 3 coding

- Object oriented to the extreme
- Write code in low level language for <u>super</u> <u>instruction</u> <u>processor</u> to obtain optimal performance
  - Fortran, C, C++
  - Non blocking MPI
  - Asynchronous I/O

ACES 3 coding

- Write algorithm in high level super instruction assembly language
  - Declare (block) arrays, (block) indices
  - DO - END DO construct
  - PARDO – END PARDO construct
  - Basic operations: add and multiply and contract
  - Each line maps to a few super instructions

Optimize and tune ACES 3

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

Outline of the talk

- What does ACES 3 do?
  - Computational chemistry
- How does it work in parallel?
  - Computer science and engineering
- **Some examples**
  - **Performance analysis**

Some tests

- Spin unrestricted SCF and CCSD
  - H_2O 115 functions, 5 occupied
  - CH_2F_2 116 functions 13 occupied
  - $C_6H_4F_2$ 140 functions 29 occupied
  - Ar_4 200 functions 36 occupied
  - Ar_6 300 functions 54 occupied
  - Ar_{10} 500 functions 90 occupied

Water

	Distrib AO	Distrib MO	Served AO	Served MO	Serial MO
Integral transform	159 1	1,977 6/2	158 1	2,307 2/2	829
Total w/o SCF	3,022 1	3,500 8	3,330 1	12,258 2/2	3,257

CH_2F_2

Segment 25
Segment 22

	Distr AO	Distr MO	Served AO	Served MO	serial
Integral transform	323 1	5,204 4,879 3/1	298 1	1,745 1,904 1/1	1,201
Total w/o SCF	12303 1	11,777 10,430 3/1	13,719 1	23,813 14,540 1/1	17,657

$C_6H_4F_2$

CCSD MO	15,856 12 1.	7,743 32 .76	4,848 64 .61
CCSD AO	35,278 8 1.	10,687 32 .82	6,294 64 .70
CCSD Geom 3 steps	255,976 12 1.	211,564 16 .91	140,424 32 .68

Ar_6 54+246=300 bf on 64 processors

Machine	SCF	trans	CCSD 1 iteration
IBM P4 shelton	313 s	4,242 s	16,363 s 4.5 h
Cray X1 diamond	582 s	6,452 s	19,601 s 5.4 h
Compaq emerald	132 s	4,180 s	29,188 s 8.1 h

Conclusion

- New design and team work delivered
- On time, within budget
- New code is fast and flexible
- New code provide new tool to do new chemistry that cannot be done without it...