Fock matrix construction

$$F_{ij} = H_{ij}^{core} + \sum_{kl} D_{kl} \left(2(kl|ij) - (ik|jl)\right)$$

- Computation is straightforward if the integrals (kl|ij) are available, but these integrals must be computed in irregularly-sized blocks called shell quartets
  - Some integrals are small and don't need to be computed
  - Don't recompute symmetric entries
- “Integral-centric” rather than “Fock matrix-centric”
- How to partition the integrals for distributed computing?

Static partitioning vs dynamic scheduling

- Static partitioning. Each part is assigned to a node. For load balance each part must require the same computational effort.
- Dynamic scheduling (task-based parallelism). Tasks are added to a queue. Nodes take tasks from the queue when they are free. Load balance is accomplished naturally.
  - Important strategy if tasks can be created by other tasks.
  - In our application, each task is 1) the computation of one (or more) shell quartets, and 2) update six blocks of F

Dynamic scheduling implementations

- Task queue is a contention point. For distributed task queues, overhead can be large.

- Hierarchical dynamic scheduling. Sets of nodes share a task queue.
- Work stealing scheduling. Each node has a queue, and “steals” tasks from other queues when the node is free.

New ideas

- For a node to compute a block of F, it needs blocks of D
    - Dynamic scheduler should be aware of data locality, to schedule tasks on nodes that already have intermediate data (blocks of D)
- If a node has a static partition, it can figure out all the blocks of D needed. Find a static partition with the following properties:
    - Relatively balanced (use task stealing to improve balance)
    - Each partition uses/reuses small portion of D

Shared-memory programming style for distributed memory system

- Each node knows what blocks of F to update. Inconvenient to have to specify which nodes owns these blocks, i.e., what data goes to which nodes
- In shared memory programming, we just “assign” to matrix locations.
- Global Arrays: the F matrix is physically distributed but logically shared. Each node updates a block of F. One-sided communication happens under the hood.

Quantum chemistry on large clusters

- Large-scale chemistry calculations on Tianhe-2

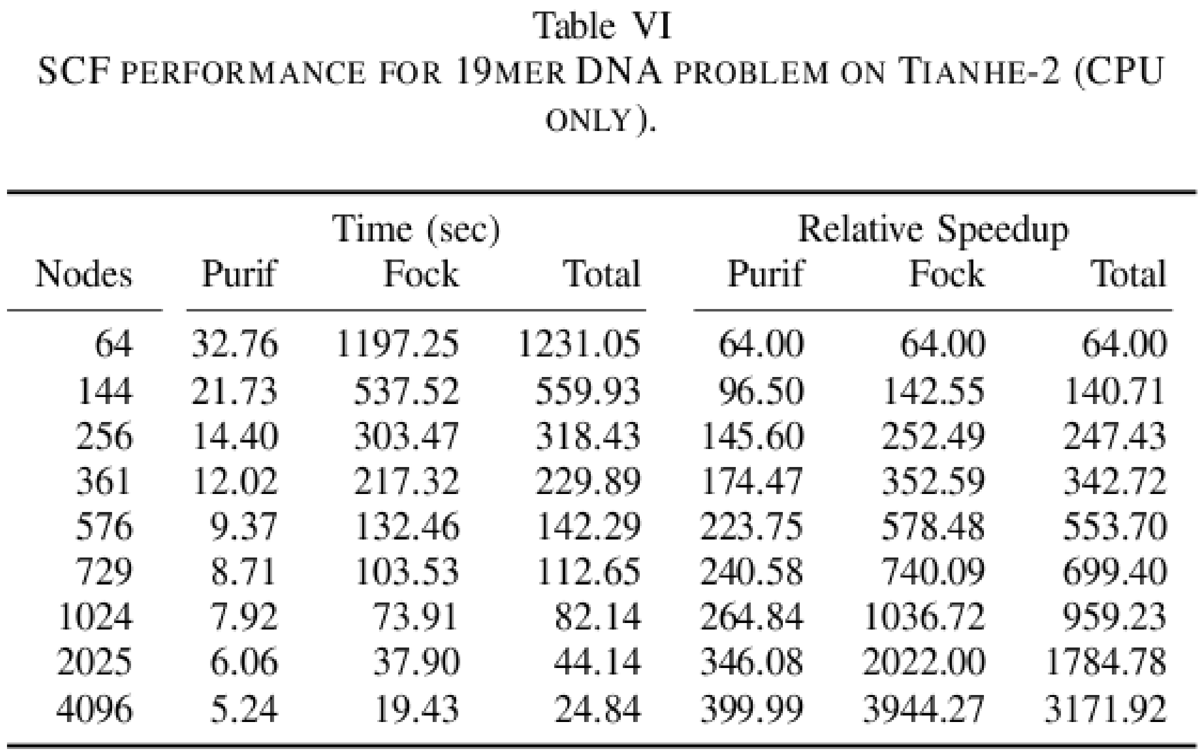

Table VI
SCF PERFORMANCE FOR 19MER DNA PROBLEM ON TIANHE-2 (CPU ONLY).

	Time (sec)			Relative Speedup		
Nodes	Purif	Fock	Total	Purif	Fock	Total
64	32.76	1197.25	1231.05	64.00	64.00	64.00
144	21.73	537.52	559.93	96.50	142.55	140.71
256	14.40	303.47	318.43	145.60	252.49	247.43
361	12.02	217.32	229.89	174.47	352.59	342.72
576	9.37	132.46	142.29	223.75	578.48	553.70
729	8.71	103.53	112.65	240.58	740.09	699.40
1024	7.92	73.91	82.14	264.84	1036.72	959.23
2025	6.06	37.90	44.14	346.08	2022.00	1784.78
4096	5.24	19.43	24.84	399.99	3944.27	3171.92

Optimize single node and SIMD performance

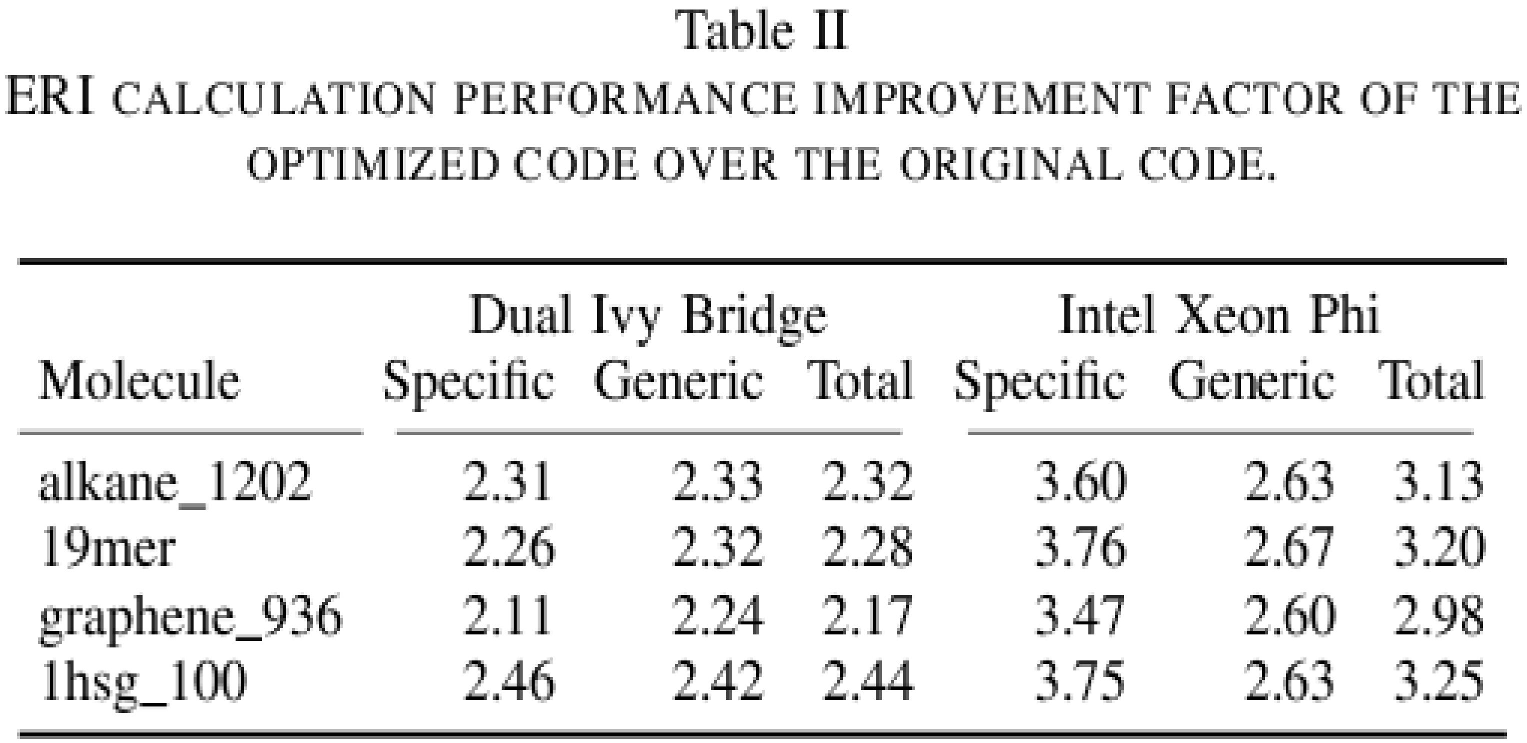

Table II
ERI CALCULATION PERFORMANCE IMPROVEMENT FACTOR OF THE OPTIMIZED CODE OVER THE ORIGINAL CODE.

	Dual Ivy Bridge			Intel Xeon Phi		
Molecule	Specific	Generic	Total	Specific	Generic	Total
alkane_1202	2.31	2.33	2.32	3.60	2.63	3.13
19mer	2.26	2.32	2.28	3.76	2.67	3.20
graphene_936	2.11	2.24	2.17	3.47	2.60	2.98
1hsg_100	2.46	2.42	2.44	3.75	2.63	3.25

Heterogeneous CPU-MIC nodes and scheduling

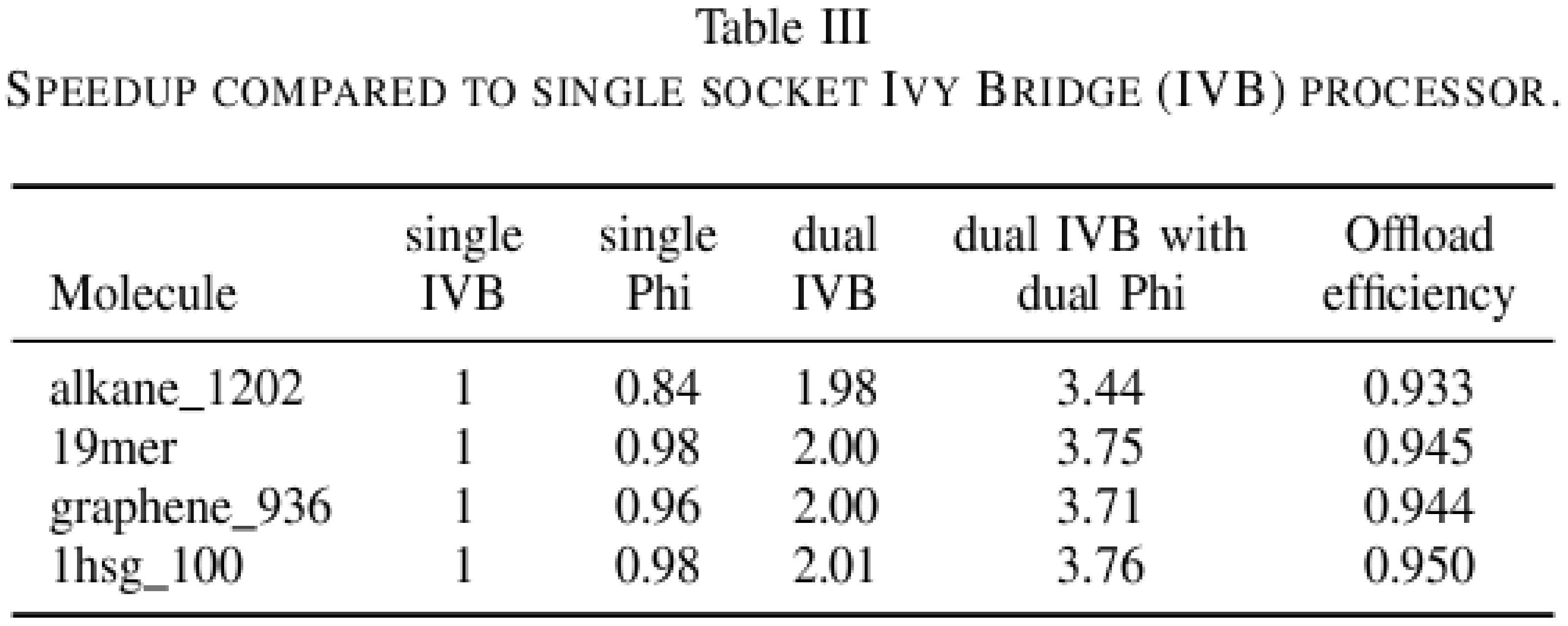

Table III
SPEEDUP COMPARED TO SINGLE SOCKET IVY BRIDGE (IVB) PROCESSOR.

Molecule	single IVB	single Phi	dual IVB	dual IVB with dual Phi	Offload efficiency
alkane_1202	1	0.84	1.98	3.44	0.933
19mer	1	0.98	2.00	3.75	0.945
graphene_936	1	0.96	2.00	3.71	0.944
1hsg_100	1	0.98	2.01	3.76	0.950

Reference: Chow et al., Scaling up Hartree-Fock calculations on Tianhe-2, Int. J. High Perf. Comput. Appl. 2016.

Tensor Contractions (Generalized matrix multiplication)

- Matrix multiply:

$$C_{ij} = \sum_k A_{ik} B_{kj}$$

- Einstein repeated index notation:

$$C_{ij} = A_{ik} B_{kj}$$

  (repeated indices on same side of equation are summed over)

Examples

- $A_{ijkl} = B_{ij}C_{kl}$
- $A_{ij} = B_iC_j$
- $A_i = B_{ij}C_j$
- $A_{ijkl} = B_{ijm}C_{mkl}$
- $A_i = B_{ijk}C_{jk}$

- Tensor contraction is associative but not commutative

Ultimate goal, in parallel…

- In computational chemistry, we will need

$$J_{ij} = D_{kl} I_{klA} I_{ijA}$$
$$K_{ij} = D_{kl} I_{ikA} I_{jlA}$$

- Try this first. Write pseudocode for

$$d_A = D_{kl} I_{klA}$$
$$J_{ij} = d_A I_{ijA}$$

- Assume all dimensions are n, and use array notation
- Consider if we first form: $I_{ikA} I_{jlA}$

Answer

$$d_A = D_{kl} I_{klA}$$

```
for A = 0..n-1
  d[A] = 0
  for k = 0..n-1
    for l = 0..n-1
      d[A] += D[k,l]*I[k,l,A]
    end
  end
end
```

Exercise

- Write code to compute J_{ij} (use random array values and n=100 or larger)
- Use OpenMP to parallelize the computation and experiment with loop scheduling
- Both steps of the computation can use BLAS-2 (dgemv, but what is the matrix and what is the vector?). Implement this version using multithreaded MKL BLAS.
- Questions
  - Timings for different loop scheduling
  - How did you use BLAS-2, timings for this version