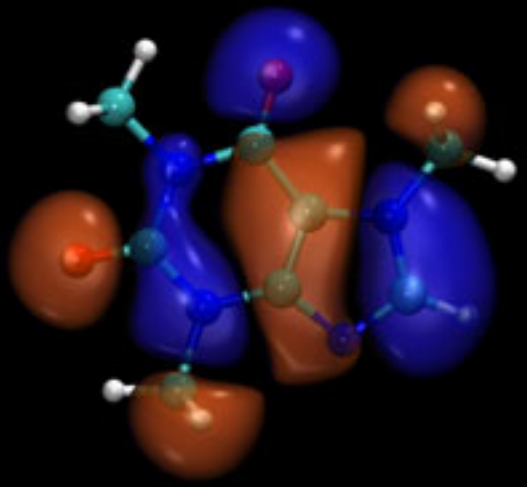
GPU-Accelerated Quantum Chemistry

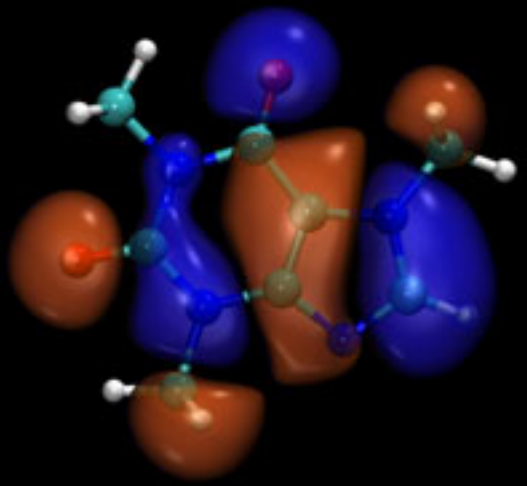
Ivan Ufimtsev
Stanford University

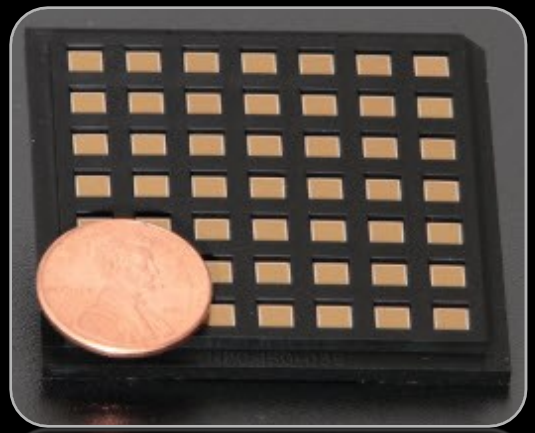


TCBG GPU Programming
Workshop, 2013

GPU (and Epiphany)-Accelerated Quantum Chemistry

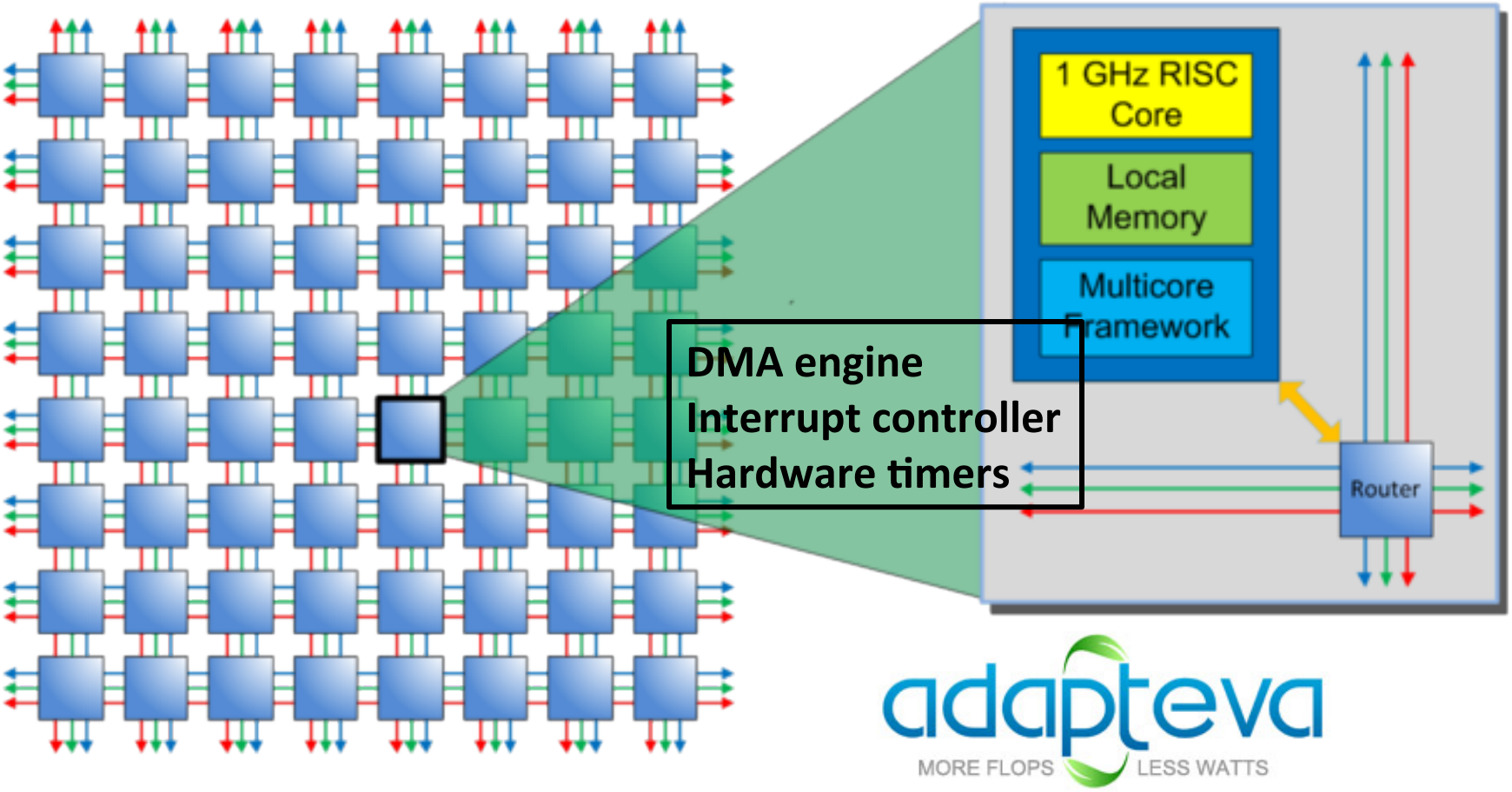
Ivan Ufimtsev





TCBG GPU Programming Workshop, 2013

The Epiphany™ Multicore Solution



adapteva
MORE FLOPS LESS WATTS

Coprocessor to ARM/Intel Host

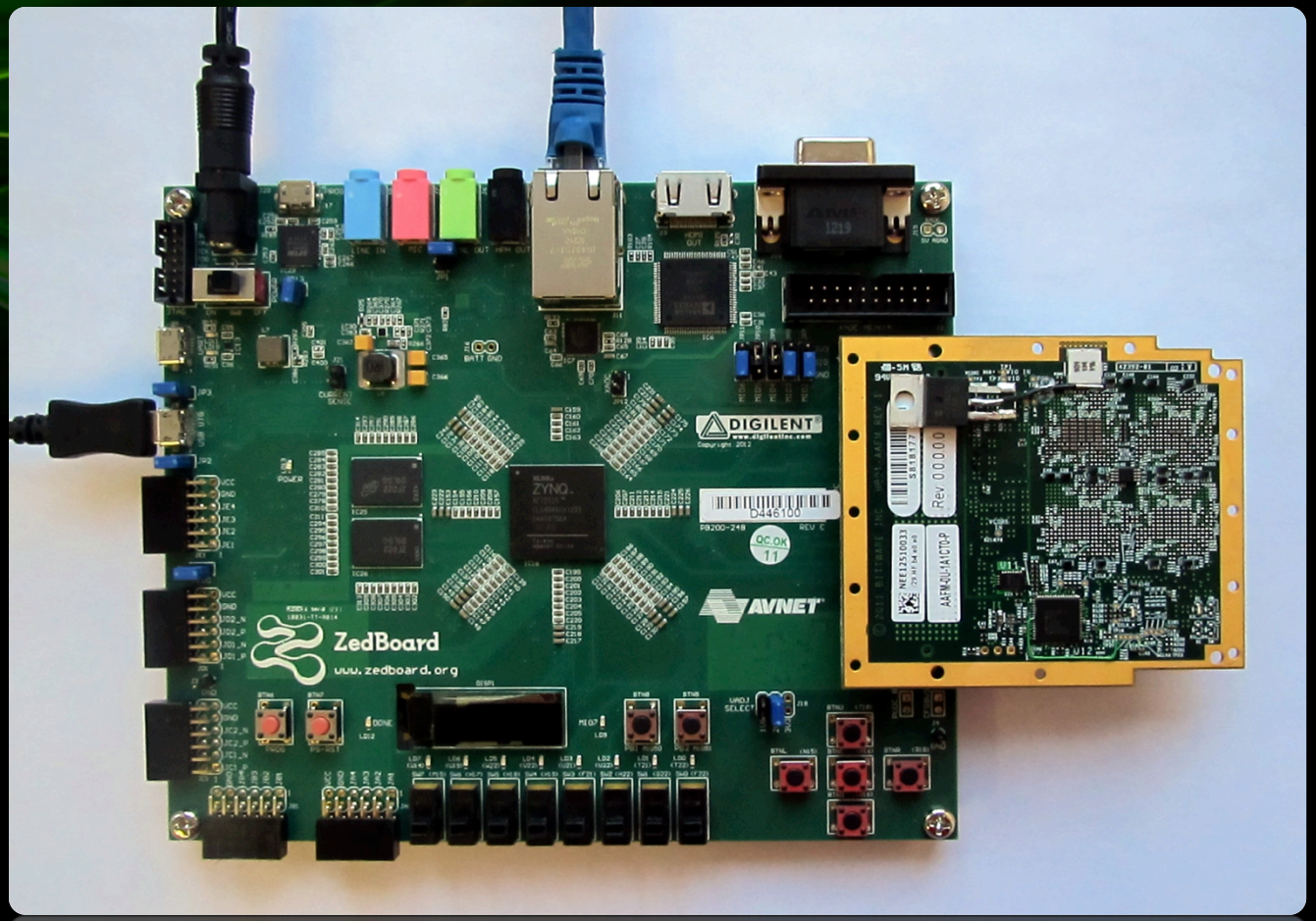
C/C++/OpenCL Programmable

Scales to 1000's of cores on a chip

16-core pre-orders/shipping to KS backers

TCBG GPU Programming
Workshop, 2013

Our 64-core prototype



TCBG GPU Programming Workshop, 2013

Our 64-core prototype

Underclocked:

600 vs 800-1000 MHz core

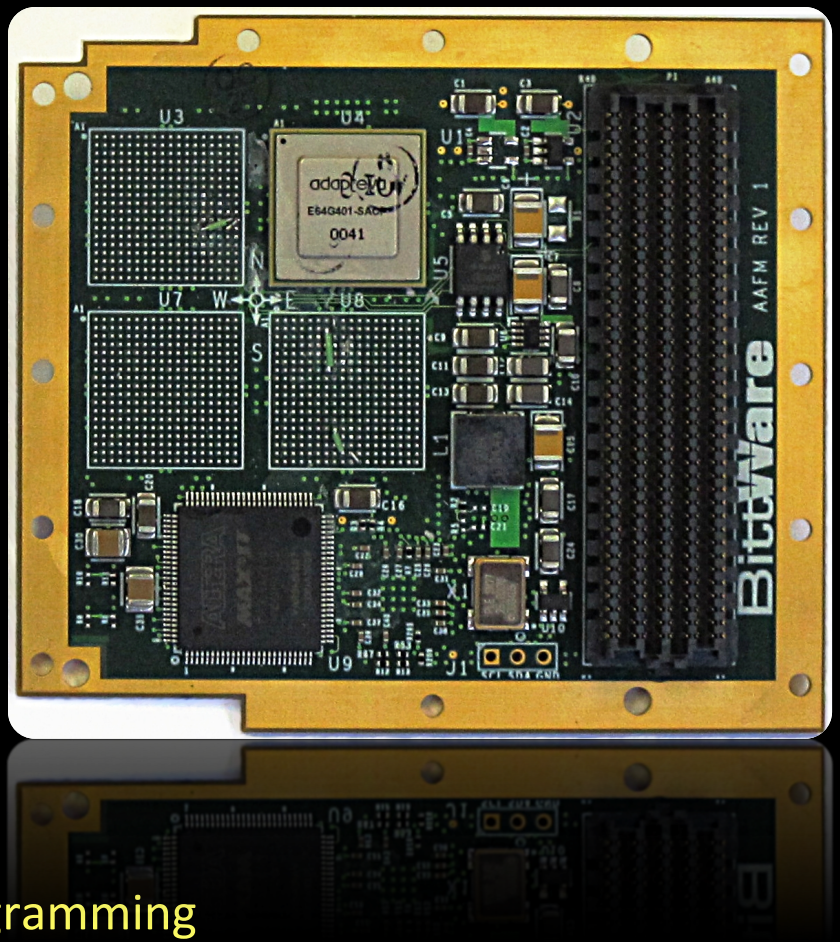
150 vs 1000's MB/s to host memory

Hardware bugs:

One of the 3 networks is broken

SDK: e-gcc based on gcc 4.8.1

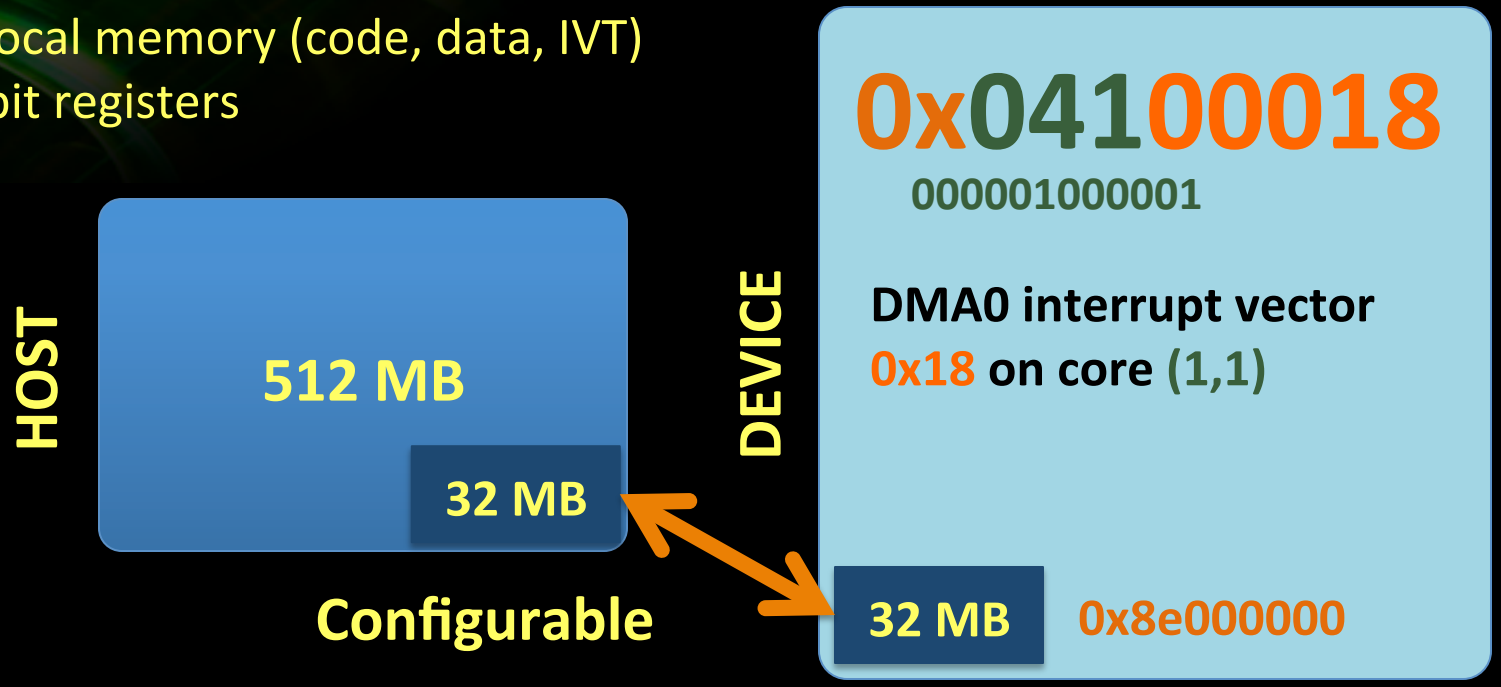
Surprisingly stable



Shared memory model

Each core's local memory, IVT, and memory-mapped registers are accessible from all other cores as well as the host

32 KB local memory (code, data, IVT)
64 32-bit registers



Multiple Program Multiple Data (MPMD)

The host and each core run their own int main() { … return 0; }

host.c

```
e_get_platform_info(&platform);
e_open(&eWorkers, 0, 0, platform.rows, platform.cols);
e_load_group((char*)"emng.srec", &eWorkers, 0, 0,          1, platform.cols, E_FALSE);
e_load_group((char*)"eint.srec", &eWorkers, 1, 0, platform.rows-1, platform.cols, E_FALSE);
```

eint.c

```
int main(void) {
  unsigned row = e_group_config.core_row - 1;
  unsigned col = e_group_config.core_col;
  unsigned rows= e_group_config.group_rows - 1;
  unsigned cols= e_group_config.group_cols;
  char* src = (char*)e_emem_config.base;
  e_dma_copy(&eInfo, src, sizeof(EInfo));
```

emng.c

```
int main(void) {
  col = e_group_config.core_col;
  cols= e_group_config.group_cols;
  EInfo  eInfo    ALIGN(8);
  src = (char*)e_emem_config.base;
  e_dma_copy(&eInfo, src, sizeof(EInfo));
  src += TO8B(sizeof(EInfo));
```

Explicit treatment of electrons is a hard problem

$$\Psi(\mathbf{r_i}) = ? \qquad E = ?$$

$$\left\{ -\frac{1}{2}\sum_i \left(\frac{\partial^2}{\partial x_i^2} + \frac{\partial^2}{\partial y_i^2} + \frac{\partial^2}{\partial z_i^2} \right) - \sum_{i,A} \frac{Z_A}{|\mathbf{r_i} - \mathbf{R_A}|} + \sum_{i,j} \frac{1}{|\mathbf{r_i} - \mathbf{r_j}|} \right\} \Psi(\mathbf{r_i}) = E\Psi(\mathbf{r_i})$$

10 electrons, 30 dimensions

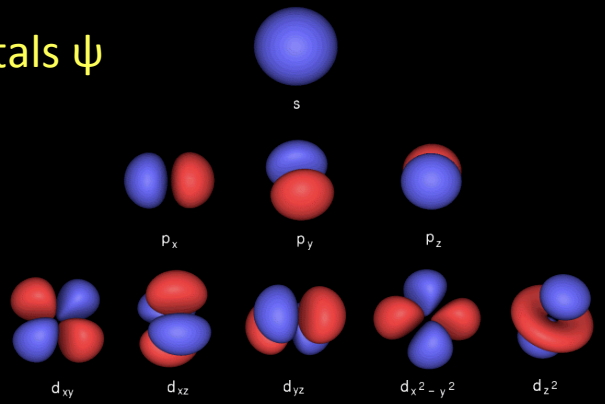
Antisymmetry: $\Psi(..,\mathbf{r_i},..,\mathbf{r_j},..) = -\Psi(..,\mathbf{r_j},..,\mathbf{r_i},..)$

3,628,000 permutations

Ψ is an antisymmetrized product of *N* 1-electron orbitals ψ

$$\Psi = A\left[\psi_1(r_1)\psi_2(r_2)...\psi_N(r_N)\right]$$

$$\psi_i(r) = \sum_{j=1}^{K} C_{ij}\varphi_j(r)$$



Self-consistent-field equations

$$F(C)C = \varepsilon C$$

F = 1e + 2e contributions: kinetic+potential "energy" of a single electron

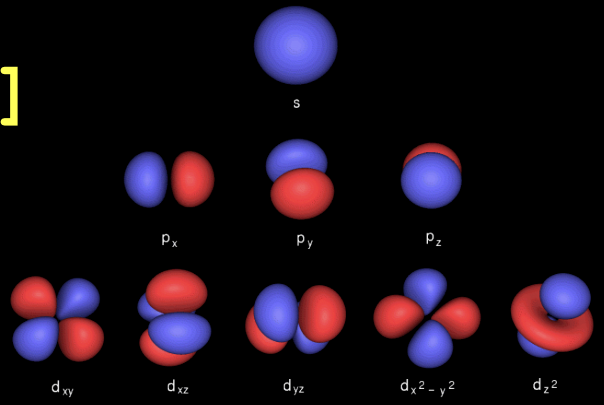
1-e term: kinetic + elec-nuc attraction + interaction with external field

2-e term: elec-elec repulsion

$$F_{ij} \mathrel{+}= \sum_{kl} [(\varphi_i \varphi_j | \varphi_k \varphi_l) D_{kl} - 0.5 (\varphi_i \varphi_k | \varphi_j \varphi_l)] D_{kl}$$

$$D_{ij} = \sum_n C_{in} C_{jn}; \quad i,j,k,l = [1 \ldots N]$$

$$(\mu\nu|\lambda\sigma) = \iint \chi_\mu(r_1)\chi_\nu(r_1)\frac{1}{|r_2 - r_1|}\chi_\lambda(r_2)\chi_\sigma(r_2)\,dr_1\,dr_2$$

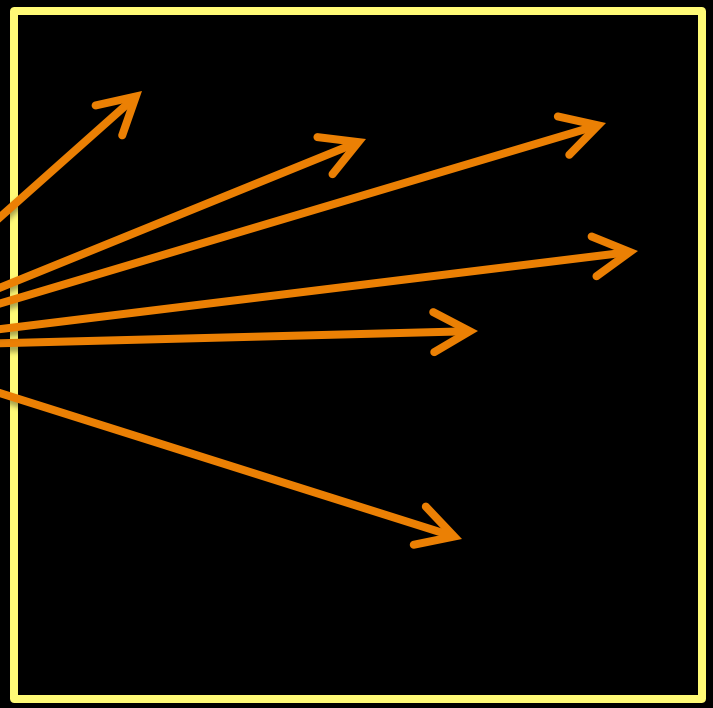


CPU: Integral symmetry and reduction

8-fold symmetry of **(ij|kl)** : $N^4/8$ integrals

ij-ji; kl-lk; ij|kl-kl|ij

Fock matrix



The curse of massive parallelism:

12 matrix (+=) updates! **(ij|kl)**

Massive read after write dependencies

Millions of mutexes? No

Atomics? Even superfast GPU atomics are not fast enough.

Not a problem for ~16 cores: replicate Fock

GPU: Compute more (redundant) integrals in exchange for local memory accesses

Compute Coulomb first, then exchange, no integral reusing

Ignore **ij|kl-kl|ij**: $2N^4/8$ integrals

$$F_{ij} \mathrel{+}= \sum_{kl} (\varphi_i \varphi_j | \varphi_k \varphi_l) D_{kl}$$

Ignore **ij-ji** and **kl-lk**: $4N^4/8$ integrals

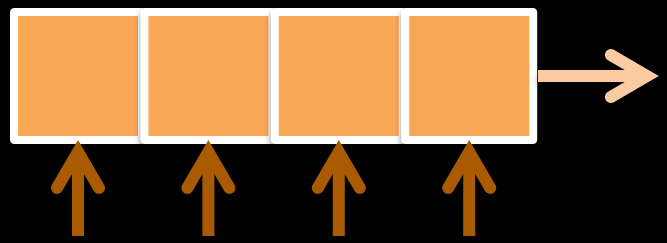
$$F_{ij} \mathrel{-}= \sum_{kl} 0.5 (\varphi_i \varphi_k | \varphi_j \varphi_l) D_{kl}$$

6 times more integrals ($6N^4/8$) and no memory collisions!

Epiphany: A smart CPU

Like a CPU, compute only non-redundant integrals:
6 times less than what a GPU has to do

"Serialize," or speaking hardware language, implement arbiters on a subset of cores

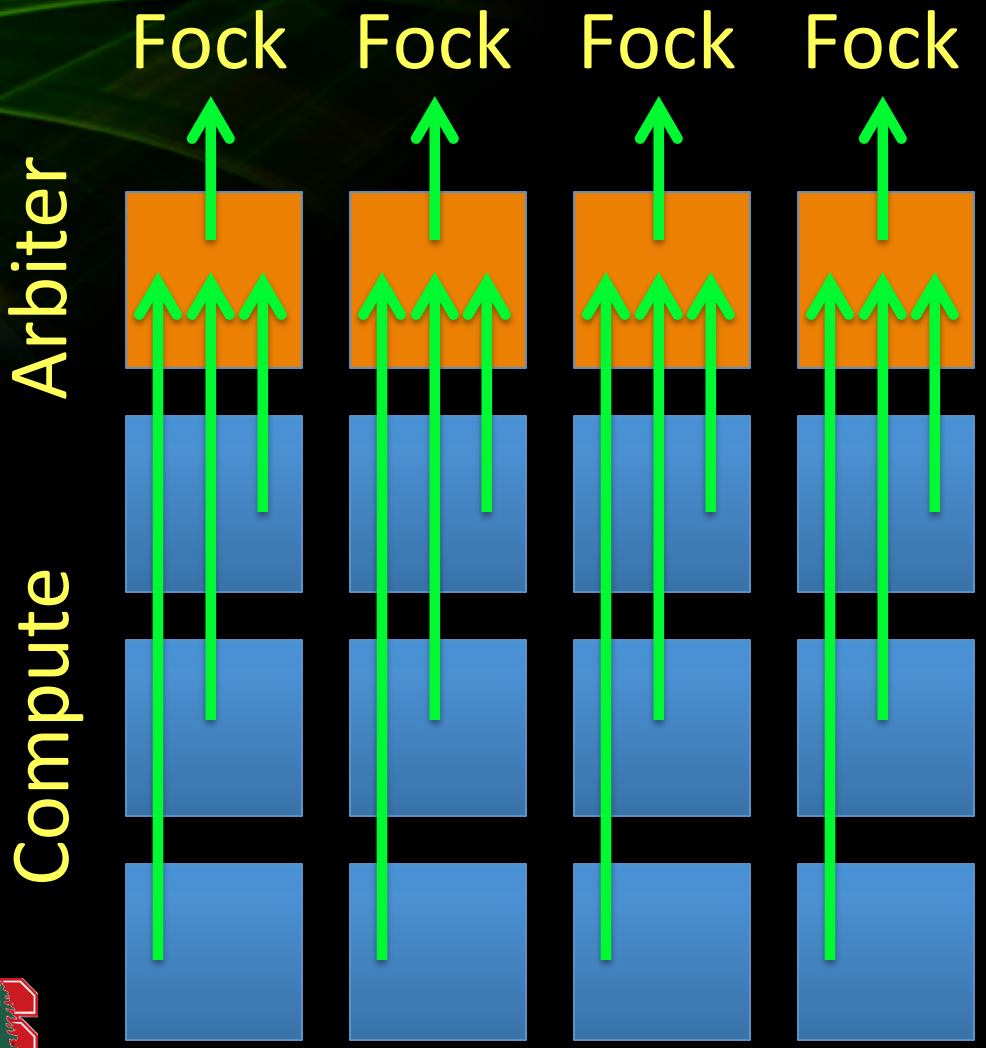


Run two programs at the same time:

1) Computes integrals and sends them to arbiters
2) Arbiters update the Fock matrix.

Both programs run at the same time on different cores

Epiphany: A smart CPU



Compute:

```
int main() {
  Loop over i, j, k, l; include all symmetries
  {
    float integral = ComputeIntegral(i,j,k,l);
    EIntegral Integral = (EIntegral){ i, j, k, l, integral };
    // remote address: 0x???04000 (data bank2), now we get rid of '?'s
    EIntegral* dst_addr = e_get_global_address(0, col, (void*)0x4000);
    e_dma_copy_msg(dst_addr + row, &Integral, sizeof(EIntegral));
  }
  return 0;
}
```

Arbiter:

```
volatile EIntegral tBuf[8] SECTION(".data_bank2"); // 0x4000

int main() {
  e_irq_attach(E_MESSAGE_INT, UpdateFock);
  e_irq_mask(E_MESSAGE_INT, E_FALSE);
  e_irq_global_mask(E_FALSE);
  while(1);
  return 0;
}

void __attribute__((interrupt)) UpdateFock(int signum) {
  Find the integral in tBuf;
  Update my own Fock;
  return; // ready to process other integrals
}
```

(ss|ss) integral timings

32 hydrogen atoms, STO-6G basis set
~ 30 FLOPS + SQRT + DIVIDE + ERFC

All emulated:	1 % of peak
erfc interpolated:	2 % of peak
inv sqrt interpolated:	12% of peak

The rest ~90%: external memory access:
150MB/s instead of 6GB/s

Conclusions

Prototypes are for prototyping, not for production runs

MPMD enables interesting experiments and non-trivial solutions

No on-chip DRAM (by design!) may be a problem

Currently one has to fit all data and code in 32KB local SRAM (s-, maybe p-, but not d-functions)

Thank you