Boyle Assembler Generation Library (BAGEL)

- State of the art multiplatform QCD asm kernel generator
- Standard tool for QCDOC kernel generation (used with CPS, Chroma for DWF, Wilson, Clover)
- Previously used on alpha clusters (e.g. UKQCD code)
- Essentially, a butt ugly domain specific compiler - looks like an abstract RISC
  - Programming interface forces user to specify the things compilers are bad at
    - Register allocation
    - Memory reference patterns
    - Loop ordering
  - (semi) Automatically does the deterministic grunt work stuff
    - Loop unrolling
    - Detailed scheduling & software pipelining
    - Prefetching/write hinting
    - Outputs multiple assembler targets (sparc, alpha, power, BG/L & “C”)
- Consists of generation & scheduling library bagel, and kernel packages bagel-wilson-dslash, bagel-qdp etc..
- Principal author is PAB. Clark & Joo have written various linear op + gamma routines
Enhanced complex interface: example (xpy)

/*Declare temporary registers*/
reg_array_1d (vec1,Cregs,3);
reg_array_1d (vec3,Cregs,3);
reg_array_1d (vec3,Cregs,3);

/*Declare mem-reference pattern (LINEAR/STRIDED/GATHER/SCATTER/LOOKUP)*/
stream *prevec1=create_stream(VEC_ATOM,vec1ptr ,counter,STREAM_IN,LINEAR);
stream *prevec2=create_stream(VEC_ATOM,vec2ptr ,counter,STREAM_IN,LINEAR);
stream *prevec3=create_stream(VEC_ATOM,vec3ptr ,counter,STREAM_IN,LINEAR);

int branch = start_loop(counter) /*Loop run by assembler*/
for(int co=0;co<3;co++) { /*Loop unrolled by code generator*/
    complex_load(vec1[co],OFFSET[co][0],vec1ptr);
    complex_load(vec2[co],OFFSET[co][0],vec2ptr);
    complex_add(vec3[co],vec2[co],vec1[co]);
    complex_store(vec3[co],OFFSET[co][0],vec3ptr);
}
prefetch_stream(prevec1); prefetch_stream(prevec2);
prefetch_stream(prevec3); /*Bagel generates prefetch patterns*/

iterate_stream(prevec1); iterate_stream(prevec2);
iterate_stream(prevec3); /*Bagel does the pointer arithmetic*/

Stop_loop(branch);
Deliberately set up an easier problem than general purpose compiler

- User decides register allocation
- User declares memory reference stream
- User decides unrolling
- C output allows rapid development and experimentation with printf debug
- Conditional logic ok in optimised loops
  - by outputting multiple versions
  - one routine can output many variants, e.g. different stream types.

- (Self selecting set of?) users are smarter than compilers
- The dslash kernels are much more complicated (e.g. stack temporaries & spill etc..)
Bagel for BlueGene

- v1.4 includes BG/L single node optimisations
  - Added complex register allocation & load store to existing complex operations interface to cover SIMD2HUMMER
  - Mixed & single precision support is work in progress

- New Wilson code generators bgl_decom, bgl_recon make use of additional registers to avoid spill to stack required on qcdoc.

- External interface to library is unmodified (so Chroma will still work).
Double Performance

• Single core performance of dslash on $4^4$
  – 1.2 Gflop/s for dslash
  – 1.1 Gflop/s 8 way decompose + 4-way SU3
  – 1.35 Gflop/s 8 reconstruct + 4 way SU3

• 8-way decompose scatters the site hopping
  – All read operands have linear access, 624 flops/site
  – $\chi_{\mu}^+ (\text{shift}(x,+\mu)) = (1+/\gamma_\mu) U_{\mu}^+(x) \psi(x)$;
  – $\chi_{\mu}^- (\text{shift}(x,-\mu)) = (1-/+\gamma_\mu) \psi(x)$;

• 8-way reconstruct is pure linear access
  – $\chi_{\mu}^{+//-}$ stored in interleaved fashion, 696 flops/site
  – $\rho(x) = \sum_{\mu} \chi_{\mu}^+(x) + U_{\mu}(x) \chi_{\mu}^-(x)$
Optimisation process

• Significant reduced load/store count vs QCDOC possible due to 64 regs vs 32.

• Initial revision showed no improvement over original QCDOC kernels (600Mflop/s).

• Reason was writes causing read/modify/write cycle through L1 cache.
  – QCDOC uses store-without-allocate MMU option, BG/L does not and causes poor performance on non-linear write streams that are not prefetched.

• Issue DCBZ (data-cache-block-zero) instructions on write streams to eliminate L1 write miss.
  – Serialising instruction causes pipeline drain and is suboptimal solution.
  – Hurts decompose performance

• Want access to an optional MMU state that store gathers instead of RMW.
  – This will then use the beautifully effective store-gathering hardware in 440 just as in QCDOC (bit like SSE’s non temporal store movntd, but automatic).

• Expect decompose to increase to 1.4-1.5 Gflop/s.
  – In discussions with IBM Yorktown over experimental kernel version with this option enabled.

• In QCDOC transient mapping of streaming data was best.
  – In discussions with IBM Yorktown over access to similar kernel experiments for transient optimisations.
Plans/ToDoList

• Mixed precision & single precision support under way - faster due to memory bandwidth issues
  – Rounding 2-spinors to single but rest of calc in double will give big performance gain. Will become a run-time option to enable rounded MD but full double accept/reject.

• Need to implement better comms & think about virtual node mode versus co-routines.
• Longer term using second (and 3rd/4th on BG/P) core via co-routines and self managed coherency will be better.

• Message sizes bigger, and no division of memory per MPI task.

• QMP over LAPI instead of MPI? Ask IBM

• True 5d DWF support (memory bandwidth vs. 2spinor footprint issues).