The HammerBlade RISC-V Manycore

A Programmable, Scalable RISC-V Fabric

Scott Davidson, Seyed Borna Ehsani, Paul Gao, Emily Furst, Tommy Jung, Sasha Krassovsky,
Max Ruttenberg, Bandhav Veluri, Leonard Xiang,
Dustin Richmond, Shaolin Xie, Chun Zhao,
Mark Oskin, **Michael Taylor**

Bespoke Silicon Group
University of Washington (http://bjump.org/manycore)

Two Key Trends

Hardware is entering an ***open source renaissance***,
e.g. open source ISAs, CAD tools, processors, libraries …

New application domains enabled not by Moore's Law but by:

new DSLs (domain specific languages) → make parallel compilation tractable

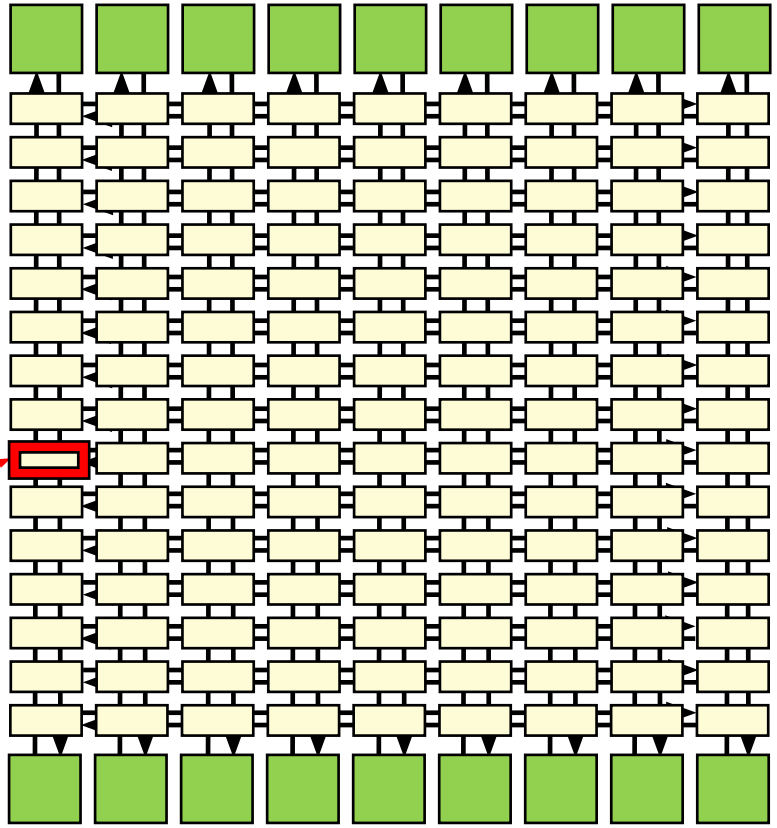
+

new parallel compute fabrics → attain energy efficiency

HammerBlade seeks to be the "base class" for these parallel compute fabrics.

HammerBlade Manycore High Level Architecture: Compute

Highly programmable, highly energy efficient parallel spatial fabric for mixed sparse/dense compute



Ultra high efficiency compute tile

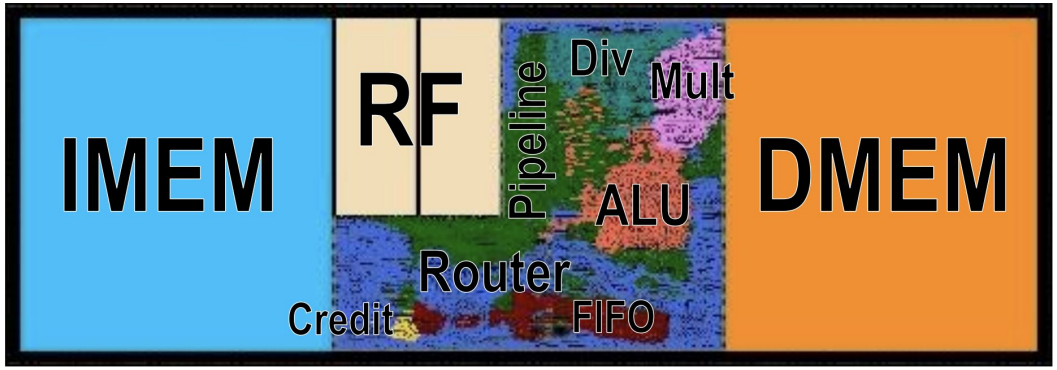
- 1 instr/cycle RISC-V engine
- >= 4 KB I-Cache
- >= 4 KB Local Data Scratchpad
- FPU
- NOC router
- Scalable, stamp out as many as you want

HB Manycore Compute Tile Has Provably Excellent Efficiency

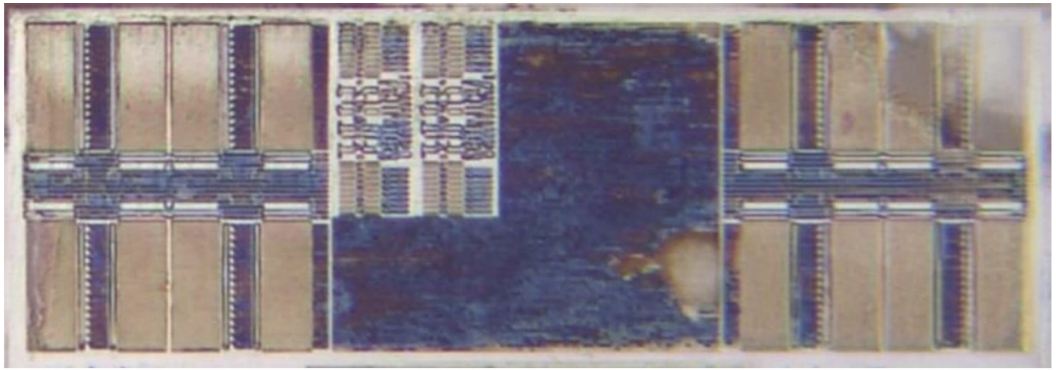
4K IMEM, 4K DMEM and 32-entry register file comprises 64% of tile area.

→ Any improvements to the tile design could at most reduce area by 36%.

40 tiles per mm^2 in 16nm!
120 tiles per mm^2 in 7nm!



Tile Floorplan in TSMC 16nm



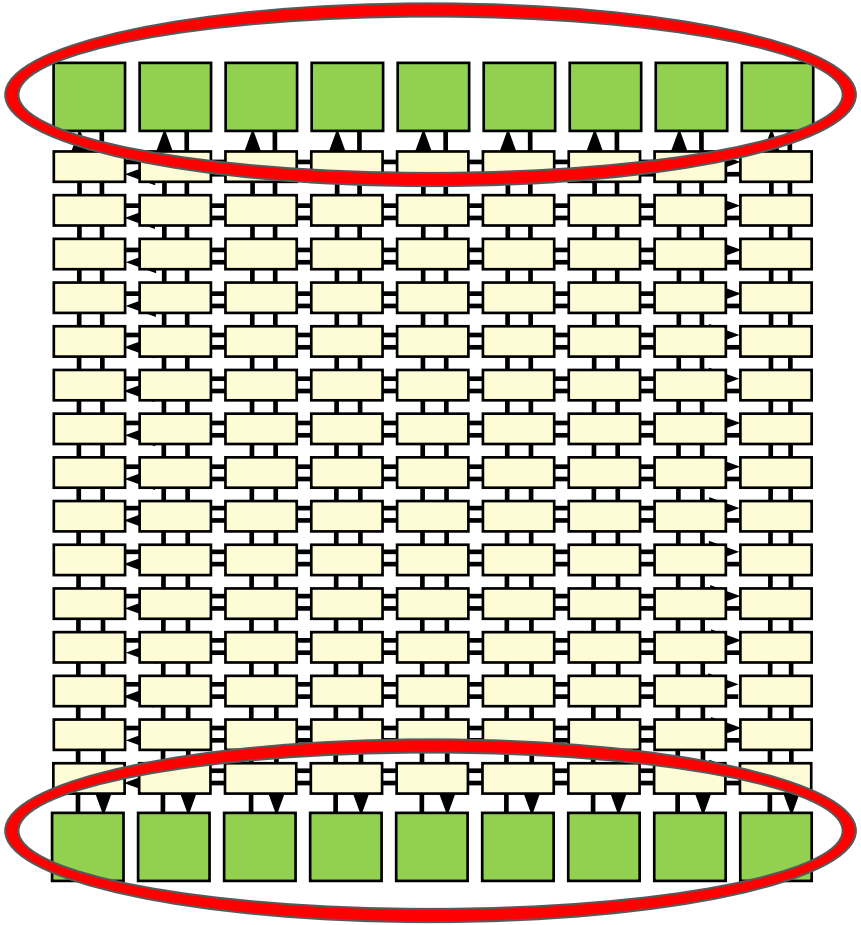
Tile Die Photo in TSMC 16nm

Manycore High Level Architecture: Global Memory

Highly programmable, highly energy efficient parallel spatial fabric for mixed sparse/dense compute

Many Parallel DRAM channels
- E.g., HBM2, DDR5, GDDR6

L2 Victim caches on each column
- Sit in front of DRAM channels
- Non-blocking
- Adapts at runtime to evolving data

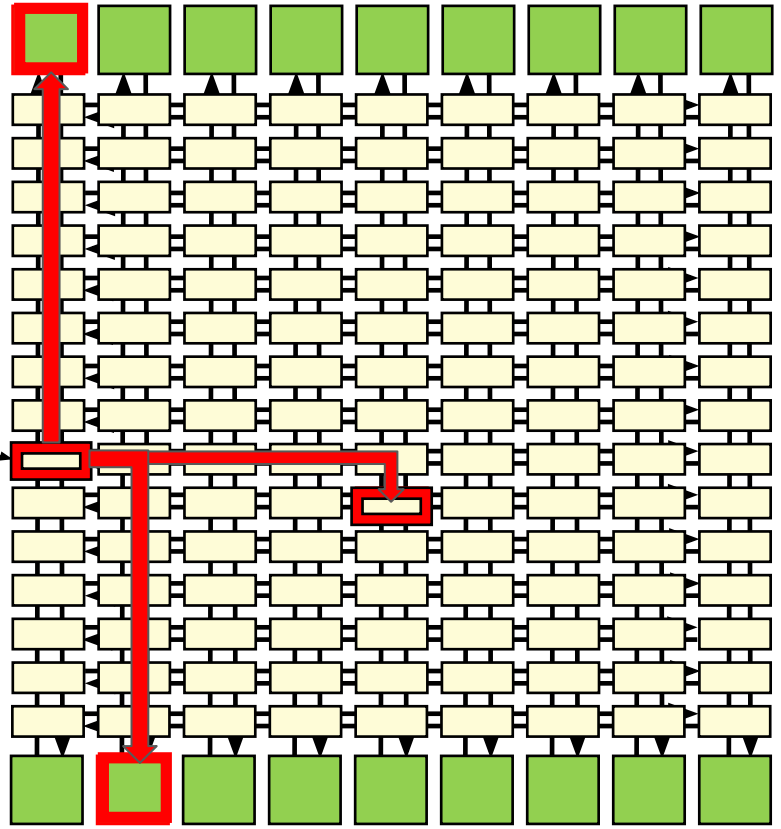


Manycore High Level Architecture: PGAS

Highly programmable, highly energy efficient parallel spatial fabric for mixed sparse/dense compute

Partitioned Global Address Space

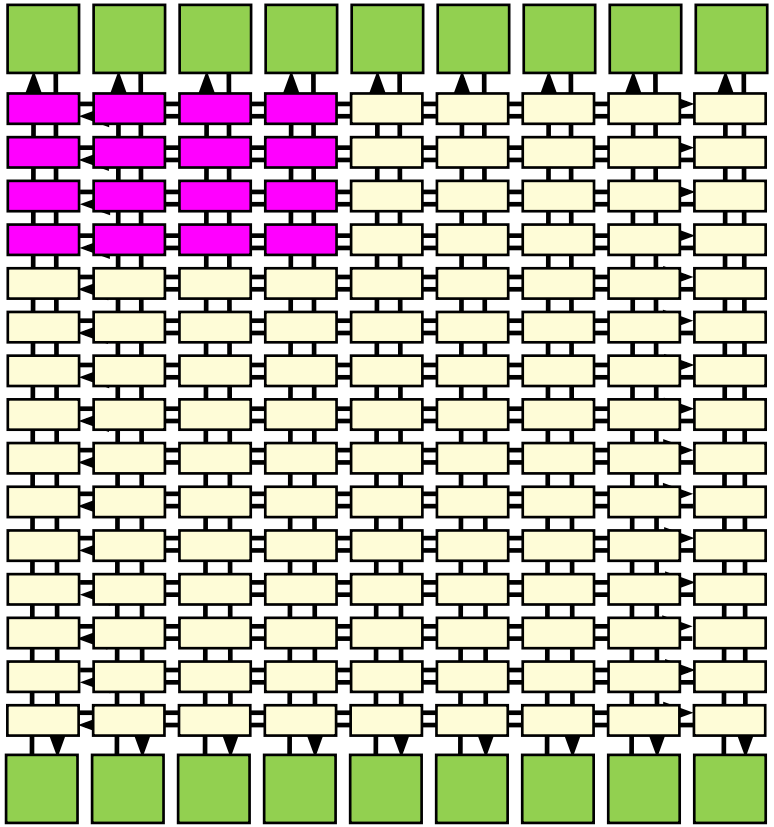
- Single LD/ST instruction to access any memory location on the chip
  - Other tiles' scratchpads
  - Global memory
- Non-blocking; each tile can have many concurrent loads and stores



Manycore High Level Architecture: Tile Groups

Tile Groups

- A Kernel is scheduled to a contiguous array of tiles, called a *Tile Group*
- Data is *striped* across the tile group's tiles' memory, and the tiles collaborate in parallel on processing the data in these nearby memories
- Larger working sets, or more parallelism? → use more tiles!
- Independent tile groups can run in parallel on different parts of the array.



CUDALITE: Low-Level C/C++ Programming for Manycore

Expert-focused programming language for high-performance library development

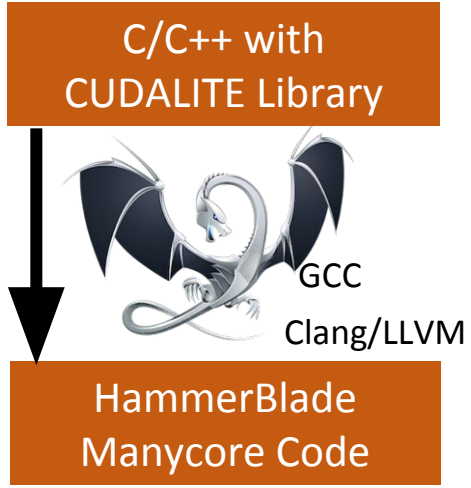
- CUDA can express independent computation and locality; widely used
- Focus on supporting same synchronization and library calls as CUDA (sync, malloc..)
- Easy to port pre-existing CUDA code over for architectural testing
- High levels of interest from industry for CUDA to RISC-V manycore

CUDA

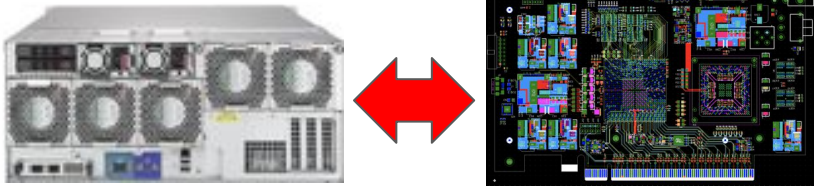
```
__global__ void add( int* a, int* b, int* c )
{
    int tid = threadIdx.x ;
    if ( tid < N )
     c[tid] = a[tid] + b[tid];
}
```

CUDALITE

```
hb_tile void add (int* a, int* b, int* c)
{
  #pragma unroll
  for ( int x = TG_Index; x < blockDim.x; x += TG_Size ) {
    c[x] = a[x] + b[x];
  }
}
```



CUDALITE Host Code:
Two kinds of hosts



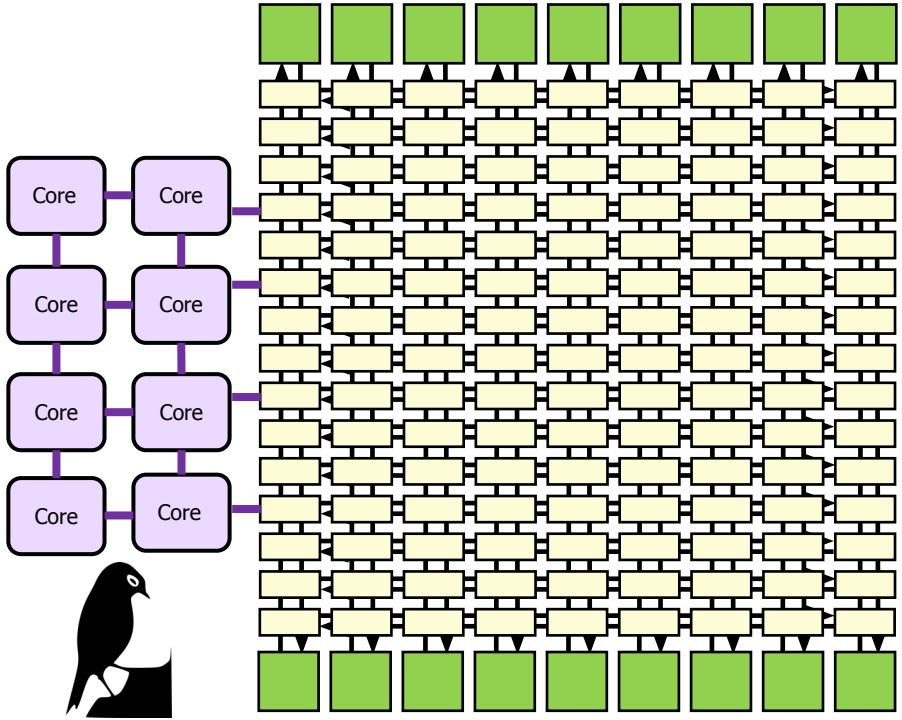
Xeon

ASIC or FPGA
Manycore
PCI-E Card

PCI-E Attached
(Leverage X86 Software!)

BlackParrot
Linux RISC-V
Multicore

HammerBlade
Manycore

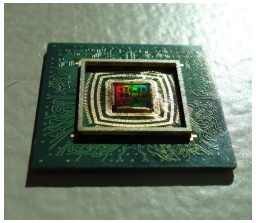


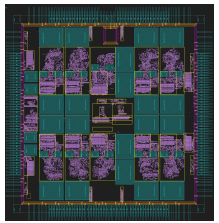
SoC Attached
(All RISC-V; Save Power)

Fast-Evolving Full Stack (HW+SW) Design (on its 4th Silicon Gen!)

V1: BSG Ten

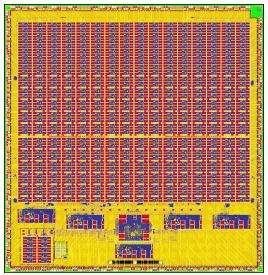
10-core system in 180nm (25 mm^2)

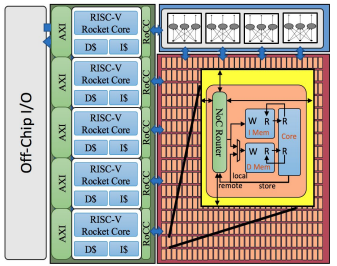




V2 & 3: Celerity

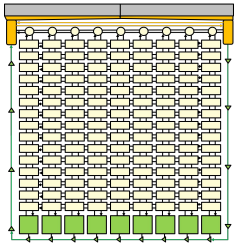
511-core system in 16nm (12 mm^2)
World record in RISC-V and Coremark perf

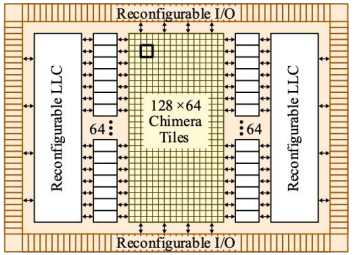




V4: HammerOne

135-core system in 12nm (6 mm^2)
Extensive programmability improvements
Floating point support



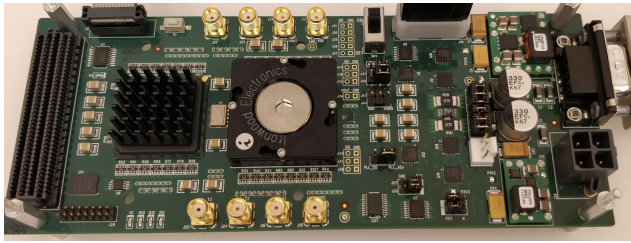


Other BaseJump Open Source Components

BaseJump STL: Standard Template Library for System Verilog

Library of high-quality implementations of almost every hardware primitive

See DAC 2018 Paper!

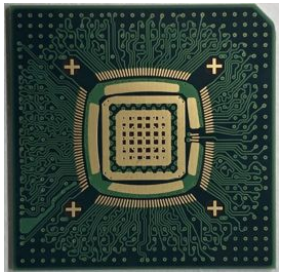


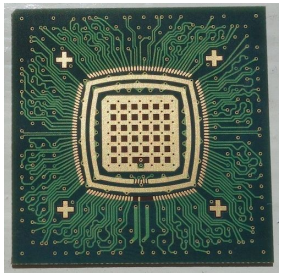
BaseJump ASIC Motherboards & Firmware

Drop your ASICs into our predesigned PCBs

BaseJump ASIC Sockets

Open Source BGA Packages & Sockets
High speed I/O over narrow links





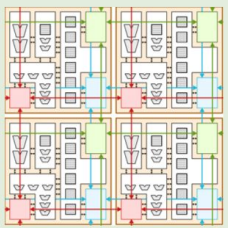
Many universities have used this to bring up their chips!

Seamless integration of new kinds of accelerators into HB manycore

(Psst .. Want to add your accelerator? We have a tutorial for you!)

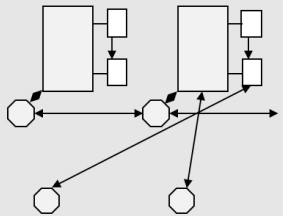
Collaborators at Cornell already adding dense and sparse matrix accelerators to HB manycore!

Dense Tensor Processing



CHIMERA

Sparse Tensor Processing

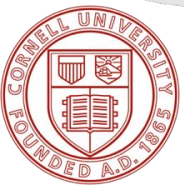


SCISSORBOX

Special Thanks To HammerBlade Cornell Team

Profs: **Adrian Sampson, Chris Batten, Zhiru Zhang**

Students & P'docs:
Philip Bedoukian
Edwin Peguero
Jie Liu
Yuewei Hu
Nitish Srivastava
Shunning Jiang
Shady Agwa
Alexa VanHattum
Neil Adit
Hanchen Jin
Zhongyuan Zhao
Peitian Pan
Yanghui Ou
Lin Cheng



HammerBlade SW Stacks

User-facing Domain Specific Frameworks We Are Developing

Drawing primarily from Graph computations, Machine Learning, and their intersection:

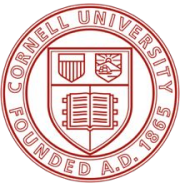
CUDALITE

GraphIt

DeepGraphLibrary

PYTORCH

tvm



GraphIt - DSL for High Performance Graph Analysis

- Decouples algorithm from optimizations
- Edge and vertex sets are the basic primitives and filter/apply operations **define the semantics of the program**
- Scheduling language controls which optimization corners are used in code generation - **allows for easy optimization space exploration**

GraphIt

```
while (frontier.getVertexSetSize() != 0)
    #s1# frontier =
        edges.from(frontier)
            .to(toFilter)
            .applyModified(updateEdge, parent, true);
end
schedule:
    program->configApplyDirection("s1", "DensePull")->generateHBCode();
```

GraphIt on HammerBlade Example

GraphIt Code

```
while (frontier.getVertexSetSize() != 0)
    #s1# frontier =
        edges.from(frontier)
            .to(toFilter)
            .applyModified(updateEdge, parent, true);
end
schedule:
    program->configApplyDirection("s1", "DensePull")->generateHBCode();
```

BFS

Generated C++ Code (Runs on x86 Host Co-processor)

```
while ( (hammerblade::builtin_getVertexSetSizeHB(frontier, edges.num_nodes()) ) != ((0) ))
{
  device->enqueueJob("edgeset_apply_pull_parallel_from_vertexset_to_filter_func",
                     {  edges.getInIndicesAddr(),
                        edges.getInNeighborsAddr(),
                        edges.num_nodes(),
                        edges.num_edges(),
                        edges.num_nodes() });
  device->runJobs();
  ...
}
...
```

GraphIt on HammerBlade Example

GraphIt Code

```
while (frontier.getVertexSetSize() != 0)
    #s1# frontier =
        edges.from(frontier)
            .to(toFilter)
            .applyModified(updateEdge, parent, true);
end
schedule:
    program->configApplyDirection("s1", "DensePull")->generateHBCode();
```

BFS

Generated C++ Code (Runs on RISC-V Manycore)

```
template <typename TO_FUNC , typename APPLY_FUNC> int
edgeset_apply_pull_serial_from_vertexset_to_filter_func_with_frontier(int *in_indices , int *in_neighbors,
TO_FUNC to_func, APPLY_FUNC apply_func, int V, int E, int block_size_x)
{
  int start, end;
  local_range(V, &start, &end);
  for ( int d=start; d < end; d++) {
    if (to_func(d) && from_vertexset[d] == 1){
      for(int s = in_indices[d]; s < in_indices[d+1]; s++) {
        if( apply_func( in_neighbors[s], d )) {
            next_frontier[d] = 1;
        }
      } //end of loop on in neighbors
    } //end of to filtering
  } //end of outer for loop
  bsg_tile_group_barrier(&r_barrier, &c_barrier);
  return 0;
} //end of edgeset apply function
```

Self-assignment of work

Parallel Dense Pull Updates

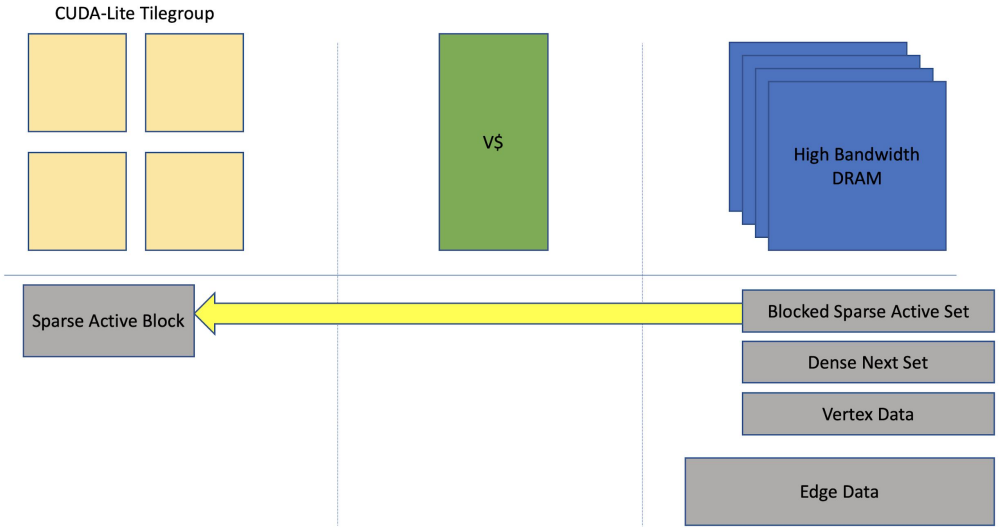
Tile group sync

GraphIt on HammerBlade Example: Blocked Structured Access

- Graph workloads are memory intensive - **taking full advantage of compute resources is a challenge**

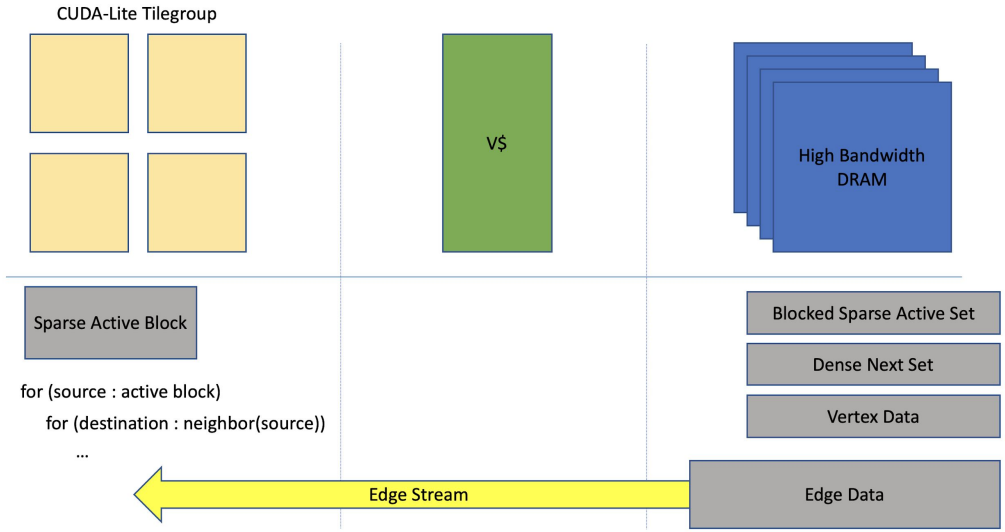
GraphIt on HammerBlade Example: Blocked Structured Access

- Graph workloads are memory intensive - **taking full advantage of compute resources is a challenge**
- Vertex data is read in small blocks into tile's local memories - **store compactly in single DRAM channel**



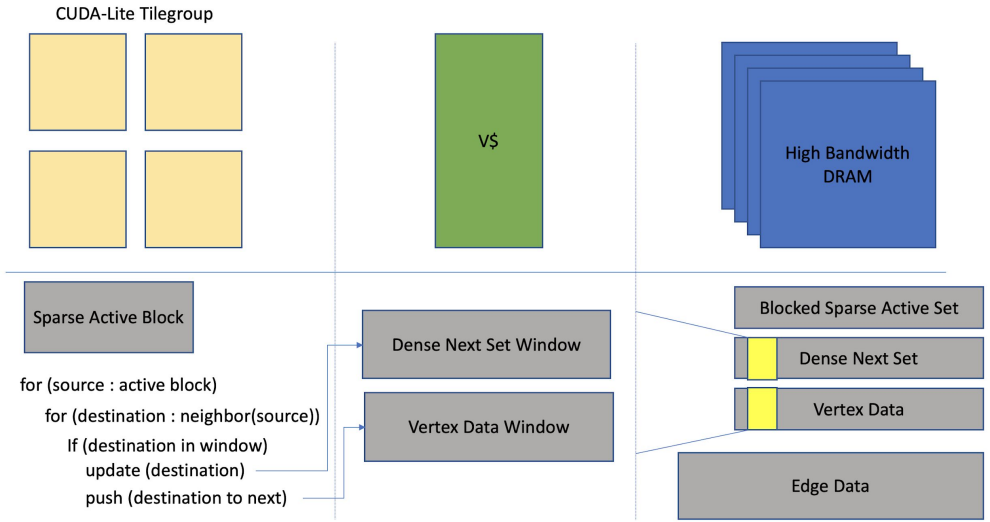
GraphIt on HammerBlade Example: Blocked Structured Access

- Graph workloads are memory intensive - **taking full advantage of compute resources is a challenge**
- Vertex data is read in small blocks into tile's local memories - **store compactly in single DRAM channel**
- Edges are partitioned across DRAM channels - **maximizes message transfer rate for sparse random access**



GraphIt on HammerBlade Example: Blocked Structured Access

- Graph workloads are memory intensive - **taking full advantage of compute resources is a challenge**
- Vertex data is read in small blocks into tile's local memories - **store compactly in single DRAM channel**
- Edges are partitioned across DRAM channels - **maximizes message transfer rate for sparse random access**
- Vertex updates are restricted to a windowed range to improve locality and prevent thrashing in caches



Getting Involved

*HammerBlade Manycore is under the **SolderPad license** (Apache 2.0 variant for HW)*

Getting Started with C/C++ Co-Simulation

Install RTL Simulator

Synopsys VCS O-2018.09-SP2

Clone the repository…

git clone git@github.com:bespoke-silicon-group/bsg_bladerunner

Get the required subprojects

git submodule init; git submodule update

Follow the instructions for running C/C++ co-simulation

https://github.com/bespoke-silicon-group/bsg_bladerunner

Open Source Verilator support would be a solid (but relatively easy) contribution from the community...

Getting Started with on Amazon F1 FPGA Instances

Install FPGA Tools

Vivado 2019.1

Clone the repository…

git clone git@github.com:bespoke-silicon-group/bsg_bladerunner

Get the required subprojects

git submodule init; git submodule update

Follow the instructions for building the FPGA and Machine images

https://github.com/bespoke-silicon-group/bsg_bladerunner#build-an-amazon-fpga-image-afi

https://github.com/bespoke-silicon-group/bsg_bladerunner#build-an-amazon-machine-image-ami

Directions you could take HammerBlade Manycore (!!)

Use & improve what we're building!

Add new SW stacks and domains:

Taco

Try out your own accelerators for these domains

Build your own FPGA or ASIC system!

The HammerBlade Team

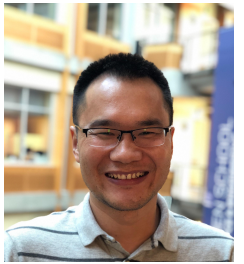
Prof. Michael Taylor

Prof. Mark Oskin

Dr. Dustin Richmond

Dr. Chun Zhao

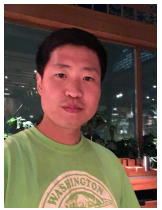
Dr. Shaolin Xie



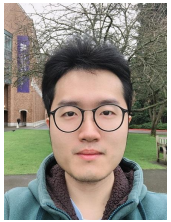
Scott Davidson

Max Ruttenberg

Tommy Jung

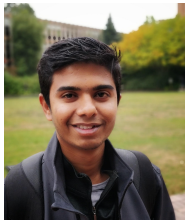


Emily Furst

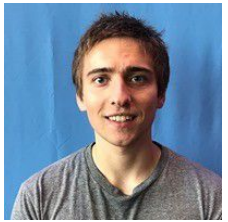


Paul Gao

Seyed Borna Ehsani

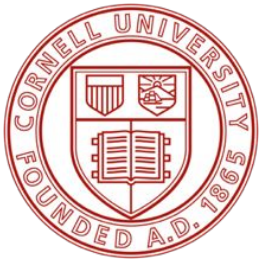


Bandhav Veluri



Sasha Krassovsky

We salute you and look forward to your contributions!



http://bjump.org/manycore

This material is based on research sponsored by Air Force Research Laboratory (AFRL) and Defense Advanced Research Projects Agency (DARPA) under agreement number FA8650-18-2-7863. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon.

DARPA

SDH