Memory Processing Units

Jaikrishnan Menon, Lorenzo De Carli, VijayraghavanThiruvengadam
Karthikeyan Sankaralingam and Cristian Estan*
UW-Madison and *Google

Today's Processors

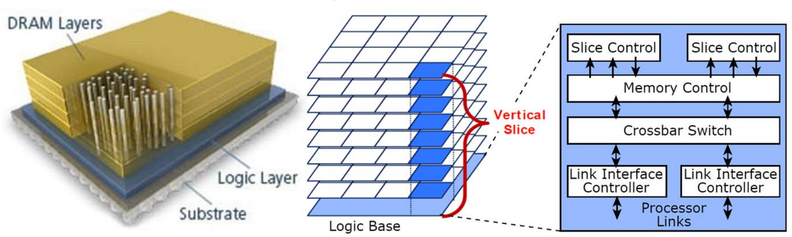
- **Cores are extraordinarily inefficient**
  - High freq., deep pipeline, speculation, caching
  - 50 pj in ALUs, 400pj in overhead
- **Memory is far away**
  - 100s of cycles
  - 500 pj to fetch 32-bit word
- Optimized for single-thread "arbitrary" programs

Opportunity 1 : Applications

- Modern applications are highly concurrent
- Offloadable
  - Have many parts that don't need high single-thread performance
- Caching doesn't do well anyway!

Opportunity 2: 3D stacked memory

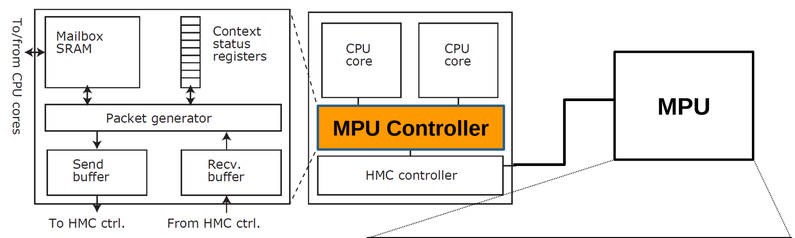
HMC (Micron & others): high bandwidth, low latency, and low power to move bits



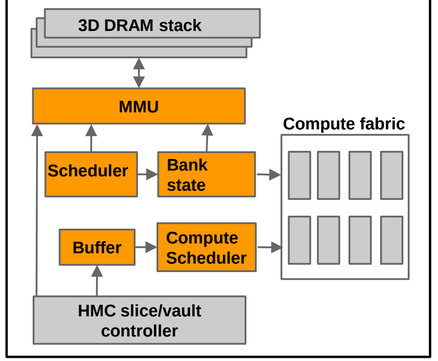
Opportunity 3: "Economics"

- Moore's law dead, Dennard scaling dead
- Conventional approaches or "traditional" technology scaling provides no improvement
- Economics of new chip design can be tolerated

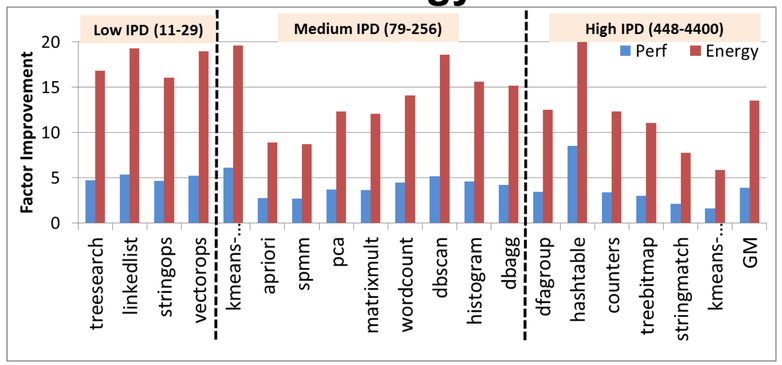
MPU hardware



- Compute fabric (90nm)
  - ARM Cortex M3
    - 250 MHz, 1-issue
    - 16KB stack, code
  - 8 cores / vault
  - 128 cores = 1.1W
- MMU can ensure sequential semantics



Performance & Energy

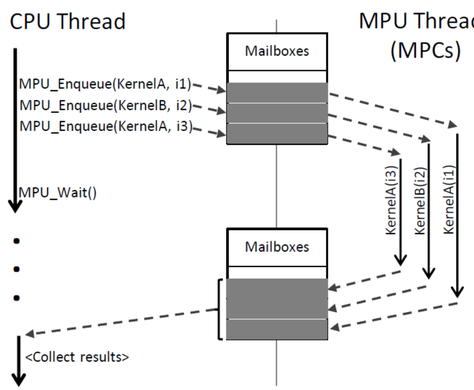


Ongoing work

- Database workloads: TPC-H Engine
- End-to-end study
- Multi-HMC system and how to get to 1TB
- FPGA prototyping & Custom cores
- **Your suggestion here**

Programming MPUs

- Shard data among DRAM chips
- CPU-to-MPU memory procedure calls start MPU threads
- MPU-to-MPU continuations



(a) Execution flow for MPU applications

```
void KVApplication() {
  MPUBox  mbox[1024];
  MPUContext ctx = MPU_CreateContext();
  MPUOpCode op1 = MPU_LoadKernel(GetValueK);
  MPUOpCode op2 = MPU_LoadKernel(SetValueK);

  for ( int c = 0, i = 0; i < N; i++ )
    if ( in[i]->type == GET )
      mbox[i] = MPU_Enqueue(ctx, op1, in[i]->data);
    else
      mbox[i] = MPU_Enqueue(ctx, op2, in[i]->data);
    if ( ++c == 1024 ) {
      MPU_Wait(ctx);
      CollectResults(mbox);
      c = 0;
    }
  }
}
```

(b) MPU-based Key-Value store

```
HTValue GetValueK(Input* i)
{
  Bucket* b = i->bucket;
  Key k = i->key;
  for ( unsigned i = 0; i < B_SIZE; i++
)
  {
    if ( b[i].key == k )
      return b[i].value;
  }

  return DEFAULT_VALUE;
}
```

```
int SetValueK(Input* i)
{
  Bucket* b = i->bucket;
  Key k = i->key;
  Value v = i->value;
  for ( unsigned i = 0; i < B_SIZE; i++ )
  {
    if ( b[i].key == k ) {
      b[i].value = v;
      return SUCCESS;
    }
  }

  return NOT_FOUND;
}
```

(c) Key-Value store application kernels