Scalable Dense Matrix Multiplication on Multi-Socket Many-Core Systems with Fast Shared Memory

Ricardo Magana, Natalia Vassilieva
Acknowledgment

Ricardo Magaña
magania@gmail.com

And also many thanks to prof. Robert Van De Geijn, Field Van Zee and Tyler Smith!
Outline

– Motivation and The Machine pitch
– NUMA-aware extension of BLIS for multi-socket systems
– Experimental results
Copper
Processor-centric computing → Memory-Driven Computing
The Machine in context

Shared nothing

Shared everything
# The Machine in context

## Communications and memory fabric

<table>
<thead>
<tr>
<th>SoC</th>
<th>Local DRAM</th>
<th>Local NVM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Shared nothing

- **SoC**
- **Local DRAM**
- **Local NVM**

## Shared something

- **Communications and memory fabric**
  - **SoC**
  - **Local DRAM**
  - **NVM**

## Shared everything

- **Physical Server**
  - **Coherent Interconnect**
  - **SoC**
  - **Local DRAM**
  - **Local NVM**

---

**Hewlett Packard Enterprise**

---

12
Our goal: efficient linear algebra library for The Machine

– Fast GEMM is crucial for fast machine learning (deep learning in particular)
– BLAS is essential for many problems in scientific computing, pattern recognition and optimization

– The ratio of compute/bandwidth on The Machine enables efficient scaling of GEMM for matrices of moderate sizes (up to 100000000 elements)
Linear algebra on The Machine: aspiration

What do we need to be true:
- High-performing single-node multi-core GEMM for small matrices
- Scalable multi-node GEMM
# Existing BLAS libraries

## Proprietary
- Intel MKL
- AMD ACML
- IBM ESSL and PESSL
- NVIDIA cuBLAS and NVBLAS

## Open Source
- ATLAS
- OpenBLAS
- BLIS
- Armadillo
- Eigen
- ScaLAPACK
- PLAPACK
- PLASMA
- DPLASMA
- Elemental
Existing BLAS libraries

**Proprietary**
- Intel MKL
- AMD ACML
- IBM ESSL and PESSL
- NVIDIA cuBLAS and NVBLAS

**Open Source**
- ATLAS
- OpenBLAS
- BLIS
- Armadillo
- Eigen
- ScaLAPACK
- PLAPACK
- PLASMA
- DPLASMA
- Elemental

**Single-node**
- Access shared coherent memory
- Threads don’t share data, only synchronization messages

**Multi-node**
- Distributed memory
- Different processes transfer data and synchronization messages

**Multi-socket with shared memory**

In The Machine we have different processes that can access shared memory
## Existing BLAS libraries

<table>
<thead>
<tr>
<th>Proprietary</th>
<th>Open Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel MKL</td>
<td>ATLAS</td>
</tr>
<tr>
<td>AMD ACML</td>
<td>OpenBLAS</td>
</tr>
<tr>
<td>IBM ESSL and PESSL</td>
<td>BLIS</td>
</tr>
<tr>
<td>NVIDIA cuBLAS and NVBLAS</td>
<td>Armadillo</td>
</tr>
<tr>
<td></td>
<td>Eigen</td>
</tr>
<tr>
<td></td>
<td>ScaLAPACK</td>
</tr>
<tr>
<td></td>
<td>PLAPACK</td>
</tr>
<tr>
<td></td>
<td>PLASMA</td>
</tr>
<tr>
<td></td>
<td>DPLASMA</td>
</tr>
<tr>
<td></td>
<td>Elemental</td>
</tr>
<tr>
<td></td>
<td>Open Source</td>
</tr>
<tr>
<td></td>
<td>Different ways of parallelization</td>
</tr>
<tr>
<td></td>
<td>Easier to optimize for a new CPU</td>
</tr>
</tbody>
</table>
Multi-socket systems today: NUMA
The ones we used

**DL580**
- 4 sockets
- 15 ivybridge/haswell cores per socket (60 cores total)
- Theoretical peak: ~2.6/5.2 TFLOPS

**Superdome X**
- 16 sockets
- 18 haswell cores per socket (288 cores total)
- Theoretical peak: ~20 TFLOPS
NUMA-aware extension of BLIS (1)
Cannon Like

- Matrix A is composed of horizontal panels
- Matrix B is composed of vertical panels
- Panels are distributed in SoC memory
- Each SoC owns one panel of A and one of B
- GEMM is distributed, each SoC computes 3 blocks, each block is obtained by panel times panel
- At every step one read from one remote SoC
- Resulting matrix have “A” format.

Node 1
Node 2
Node 3

Node 1
Node 2
Node 3
NUMA-aware extension of BLIS (2)

Blocks

• A and B have the same format
• As previous every SoC reads from only one other SoC
• Unlike previous switch reading SoC after each block.

Node 1
Node 2
Node 3

- SoC 1 Compute
- SoC 2 Compute
- SoC 3 Compute
Other tricks

- Support for different memory pools (for different panels)
  - The entry point (bli_gemm) receives an array of obj_t that represent the panels of the matrix
- MCS barrier instead of linear
- Support for multiple thread entry points
  - To do not spawn new set of threads at every iteration (in every bli_gemm call)
- Affinity of threads
  - We pre-launch the threads, pin them to particular CPU cores using a #pragma omp (outside of blis), and then use multiple threads entry points
SGEMM performance on Superdome X, comparison with a GPU system (2 NVIDIA Tesla K80)
SGEMM performance on Superdome X

DISTRIBUTED SGEMM PERFORMANCE

- nvBLAS (4 GPUs)
- nvBLAS (2 GPUs)
- nvBLAS (1 GPU no copy)
- NUMA-BLIS v1
- Custom + BLIS
- nvBLAS (1 GPU)
Improved usability and performance for small matrices (v2)
Distributed SGEMM on Superdome X
Conclusion

– Done (almost): Extended BLIS (GEMM so far…) for multi-socket systems with shared memory
  – Matrix data is accessed directly
  – Synchronization via barriers
  – NUMA-aware

– In progress: Extended BLIS for The Machine
  – Matrix data is accessed directly
  – Matrix data is in NVM
  – Synchronization via MPI/RVMA
Thank you!
nvassilieva@hpe.com