Towards Automatic HBM Allocation using LLVM: A Case Study with Knights Landing

Dounia Khaldi and Barbara Chapman
Institute for Advanced Computational Science
Stony Brook University
Stony Brook, NY

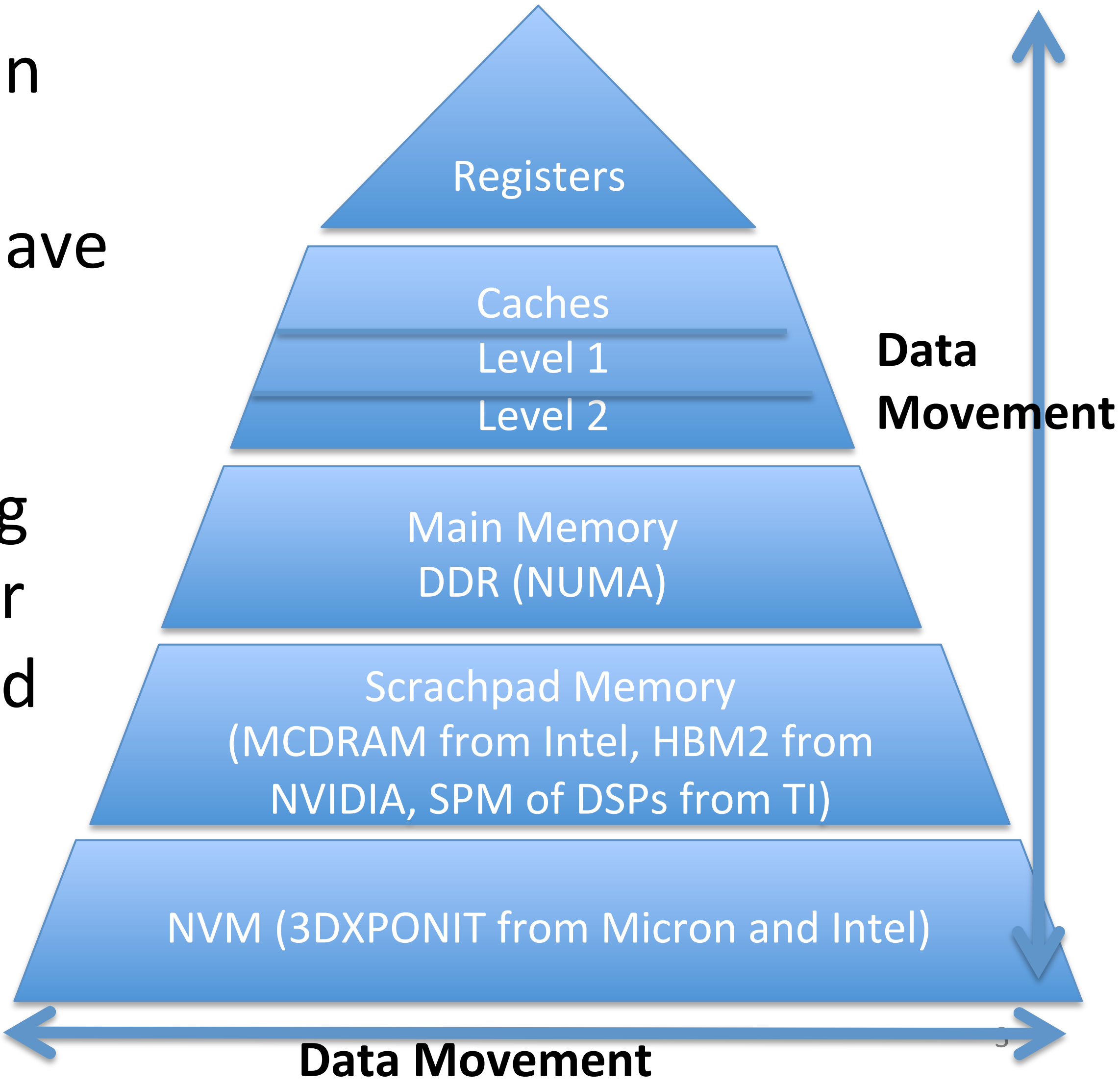
The Third Workshop on the LLVM Compiler Infrastructure in HPC
Salt Lake City, Utah, November 14, 2016

Outline

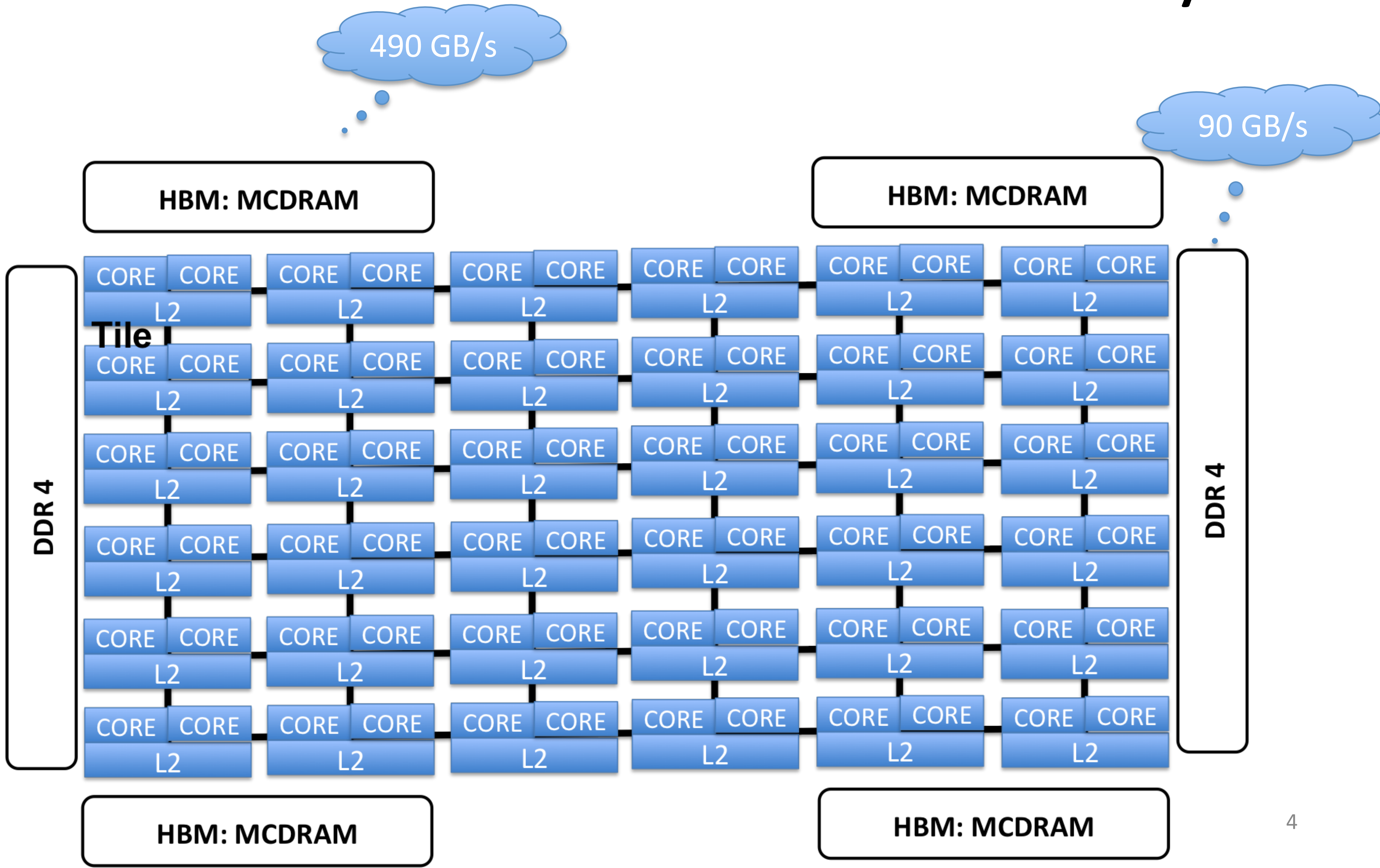
- Introduction and Motivation
- Methodology:
  - Bandwidth-Critical Data Analysis (BCDA)
  - HBM Allocation Transformation
- Experimental Results using CG benchmark
- Conclusion and Future Work

Introduction: Exploring Memory Hierarchy

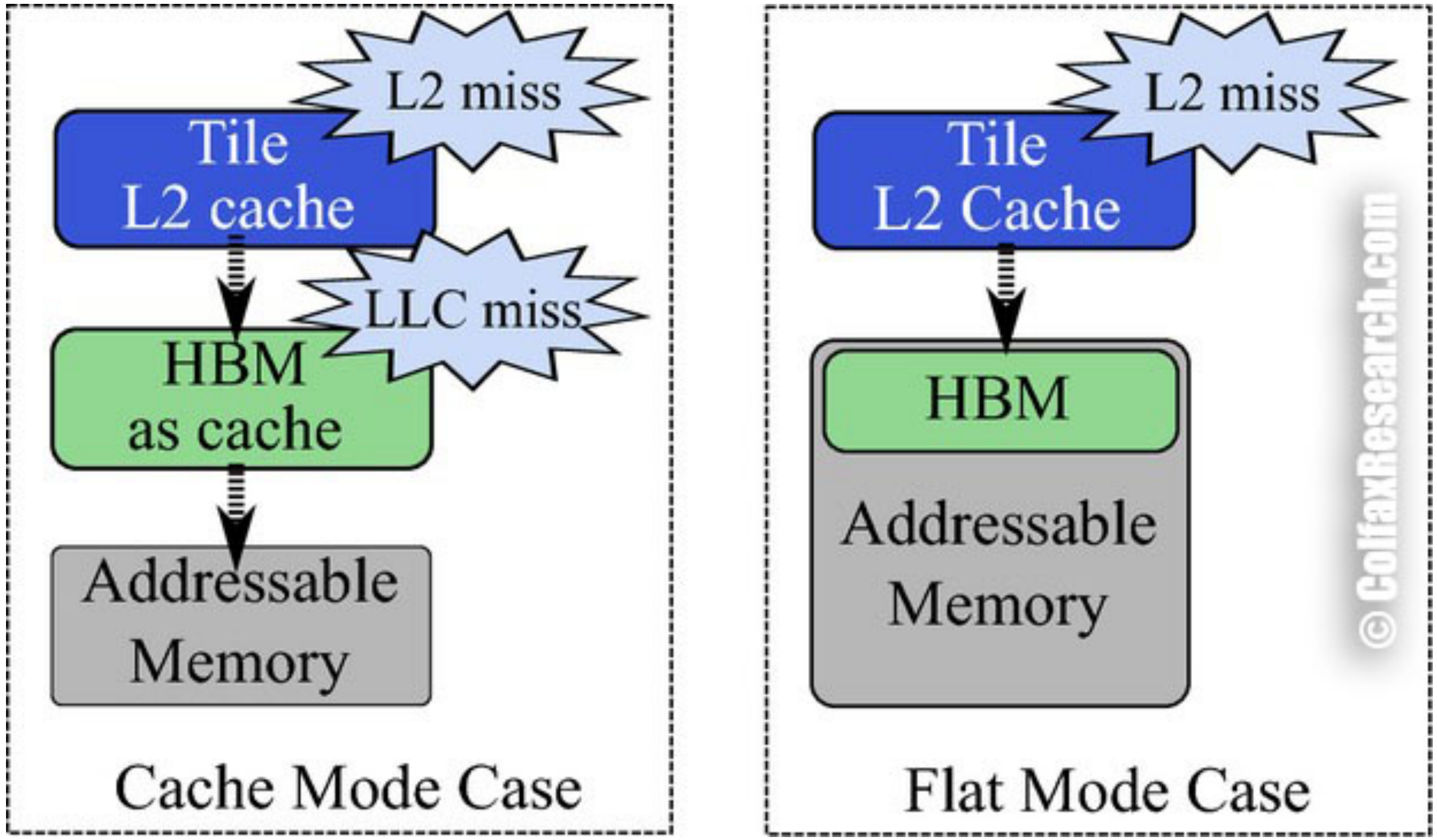
- New kinds of memory in new architectures
- Which data elements have to reside on these memories?
- High performance using HBM, with lower power requirements compared to DDR
- 3D Xpoint offers 1,000 times the performance of today's SSDs


Registers
Caches
Level 1
Level 2
Main Memory
DDR (NUMA)
Scrachpad Memory
(MCDRAM from Intel, HBM2 from NVIDIA, SPM of DSPs from TI)
NVM (3DXPONIT from Micron and Intel)
Data Movement
Data Movement

KNL Architecture as a Case Study



MCDRAM Configuration Modes

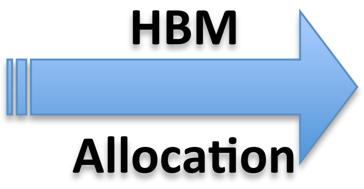


Extracted from http://colfaxresearch.com/knl-mcdram/

Programming KNL MCDRAM: Flat Mode

- hbwmalloc library

```
float *fv;
fv = (float *)
malloc(sizeof(float)*n);
```



```
float *fv;
fv = (float *)
hbw_malloc(sizeof(float)*n);
```

- Intel memkind library
  - C, C++: `memkind_malloc()`
  - Fortran:
    - `!DIR$ ATTRIBUTES FASTMEM :: object`
    - Since Intel Fortran 16.0 compiler
- AutoHBW library
  - Threshold size: `AUTO_HBW_SIZE`
- numactl command

Related Work

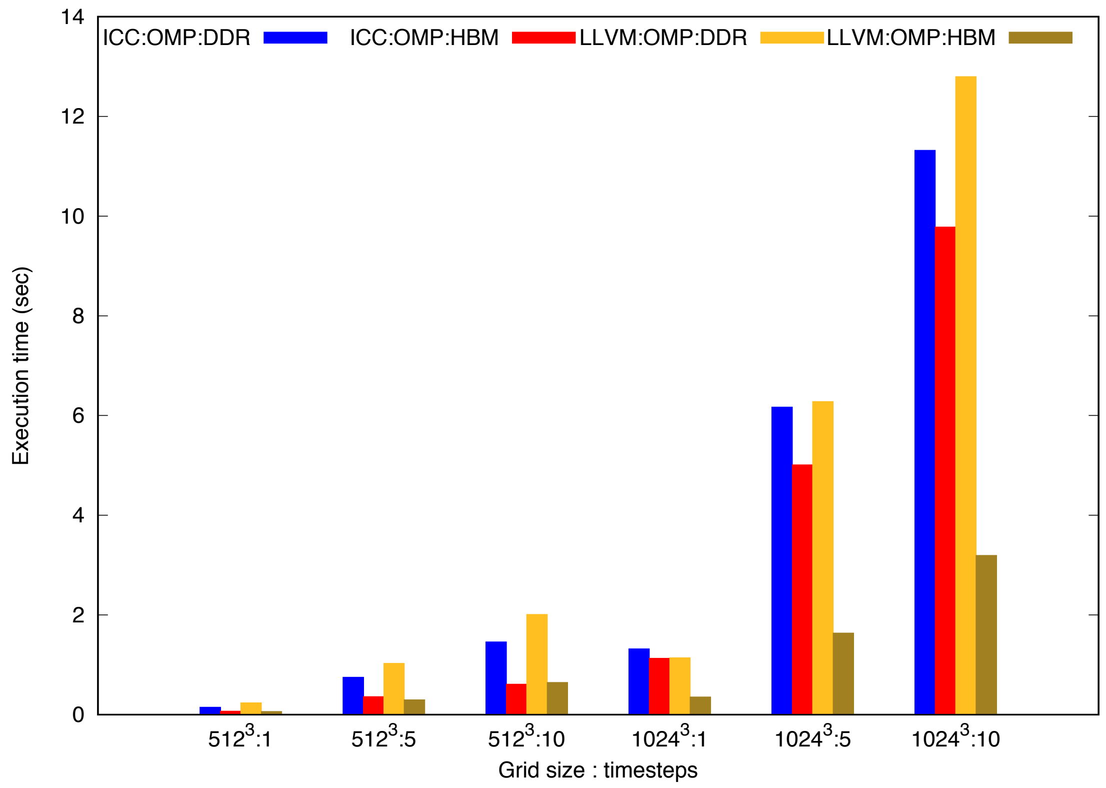
Level	Work	Example
API Level	Legion, Sequoia, RDDs, Adios	persistent
	OpenMP 5.0?	Current proposal: #pragma omp allocate With memory spaces, allocators and traits
	Vendors Low Level Libraries from Intel and Cray	Cray: #pragma memory(bandwidth)
Compiler level	Compiler transformations	Loop nests
Tools	VTune	Collect bandwidth profiles • Dynamic • bandwidth information and then what?

- We use LLVM, a widespread SSA-based compilation infrastructure for sequential and parallel languages
- Decide when it is beneficial to allocate data in the HBM for sequential and OpenMP code
- Case study: the HBM, called MCDRAM, of Knights Landing (KNL)

Level		
API Leve		
		memory spaces, allocators and traits
	Vendo Level Libraries from In el and Cray	Cray: #pragma memory(bandwidth)
Compiler level	Compiler transformations	Loop nests
Tools	VTune	Collect bandwidth profiles • Dynamic • bandwidth information and then what?

Motivation: Impact of MCDRAM on OpenMP 3D 7-point Stencil

- Setup: 1-node machine with one Intel(R) Xeon Phi(TM) CPU 7210 processor @ 1.30GHz configured with the Sub NUMA clustering mode
- ICC 16.0.3 and LLVM 3.8.1, with –O3
- DDR vs. HBM execution time of OpenMP version of 3D 7-point Stencil
- `hbw_set_policy(HBM_POLICY_BIND);`



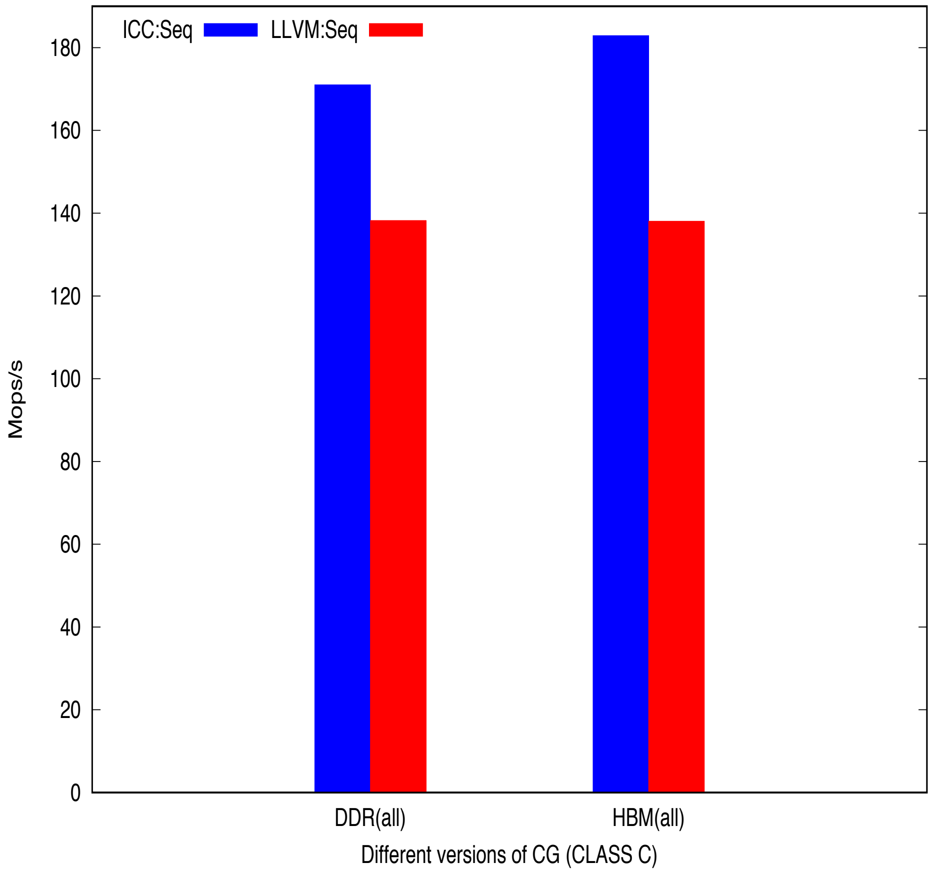
What to allocate into HBM?

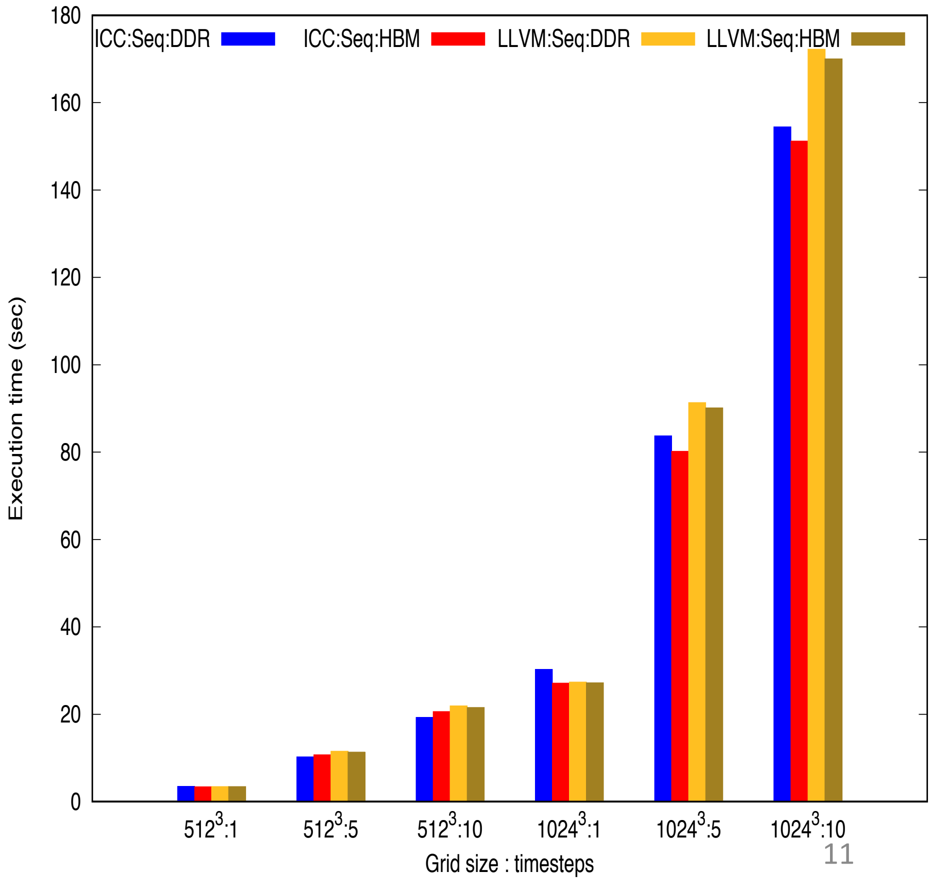
- Snippet code (NAS-NPB CG benchmark)
- Different types of memory accesses
- Several matrix and vector multiplications and additions

```
for (cgit = 1; cgit <= cgitmax; cgit++){
  ...
  #pragma omp for
  for (j = 0; j < lastrow - firstrow + 1; j++) {
    suml = 0.0;
    for (k = rowstr[j]; k < rowstr[j+1]; k++) {
      suml = suml + a[k]*p[colidx[k]];
    }
    q[j] = suml;
  }
  #pragma omp for reduction (+:d)
  for (j = 0; j < lastcol -firstcol + 1; j++){
    d = d + p[j] * q[j];
  }
  ...
}
```

Bandwidth-Critical Data (1)

- Many wires into MCDRAM → simultaneous access is needed





Bandwidth-Critical Data (2)

- Predictable memory access patterns → application is bandwidth-bound
- Many wires into MCDRAM → simultaneous access is needed
- Library solutions → not portable
- API level: might be a burden

→ Compiler + runtime solution

Methodology: Bandwidth-Critical Data Analysis (BCDA)

$$R = R(v)$$

$$P(R) = bandwidth(R)\sum_{r\in R}\cos t(r)workshare(r)$$

$$\cos t(r) = \begin{cases} 2 \text{ if is a store operation} \\ 1 \text{ otherwise} \end{cases}$$

$$workshare(r) = \begin{cases} 0 \text{ if } r \text{ is individual} \\ 1 \text{ if } r \text{ is simultaneous} \end{cases}$$

$$bandwidth(R) = \begin{cases} 1\ \forall r \in R, \text{ r is regular} \\ 0 \text{ otherwise} \end{cases}$$

BCDA: Interprocedural Memory Operations Count

$$R = R(v)$$

$$P(R) = bandwidth(R)\sum_{r \in R} \cos t(r) workshare(r)$$

- LLVM IR is in SSA form
  - *One definition → multiple uses*
  - Allows for Def-Use and Use-Def chain analysis
- Interprocedural Memory Operations Count ($\sum$)
  - `__kmpc_fork_call`
  - Number of memory operations in the generated LLVM IR (`load, store` and `getelementptr`)

BCDA: Data Reuse Cost

$$P(R) = bandwidth(R)\sum_{r \in R} \cos t(r) workshare(r)$$

$$\cos t(r) = \begin{cases} 2 \text{ if } r \text{ is a store operation} \\ 1 \text{ otherwise} \end{cases}$$

- Function *cost* assigns a weight to reference operations

BCDA: Individual vs. Simultaneous Access

$$P(R) = bandwidth(R) \sum_{r \in R} \cos t(r) workshare(r)$$

$$workshare(r) = \begin{cases} 0 \text{ if } r \text{ is individual} \\ 1 \text{ if } r \text{ is simultaneous} \end{cases}$$

- OpenMP as a case study
- Function *workshare* detects if an access *r* has been performed in an OpenMP work-sharing region or not

BCDA: Regular vs. Irregular Access Pattern

$$P(R) = bandwidth(R) \sum_{r \in R} cost(r) workshare(r)$$

$$bandwidth(R) = \begin{cases} 1 & \forall r \in R, \text{ r is regular} \\ 0 & \text{otherwise} \end{cases}$$

- Function *bandwidth:* latency vs bandwidth bound
- Indirect Accesses: indices arguments of the *getelementptr* instruction

Methodology: Allocation Transformation

```
int *a =
malloc(sizeof(int) * n);
```

```
%call3 = call i8* @malloc(i64
%mul)
%6 = bitcast i8* %call3 to
i32*
store i32* %6, i32** @a, align
8
```

```
%call31 = call i8*
@memkind_alloc(i64 %mul)
%6 = bitcast i8* %call31 to
i32*
store i32* %6, i32** @a, align
8
```

```
#if defined (HAVE_HBWMALLOC_H)
  # include <hbwmalloc.h>
void *memkind_alloc(size_t size) {
  int avail = hbw_check_available();
  void *a;
  hbw_set_policy(HBW_POLICY_PREFERRED);
  if(avail == 0){
    a = hbw_malloc(size);
    assert(a != NULL);
  }
  else{
    a = malloc(size);
  }
  return a;
}
#else
void *memkind_alloc(size_t size) {
  void *a = malloc(size);
  return a;
}
```

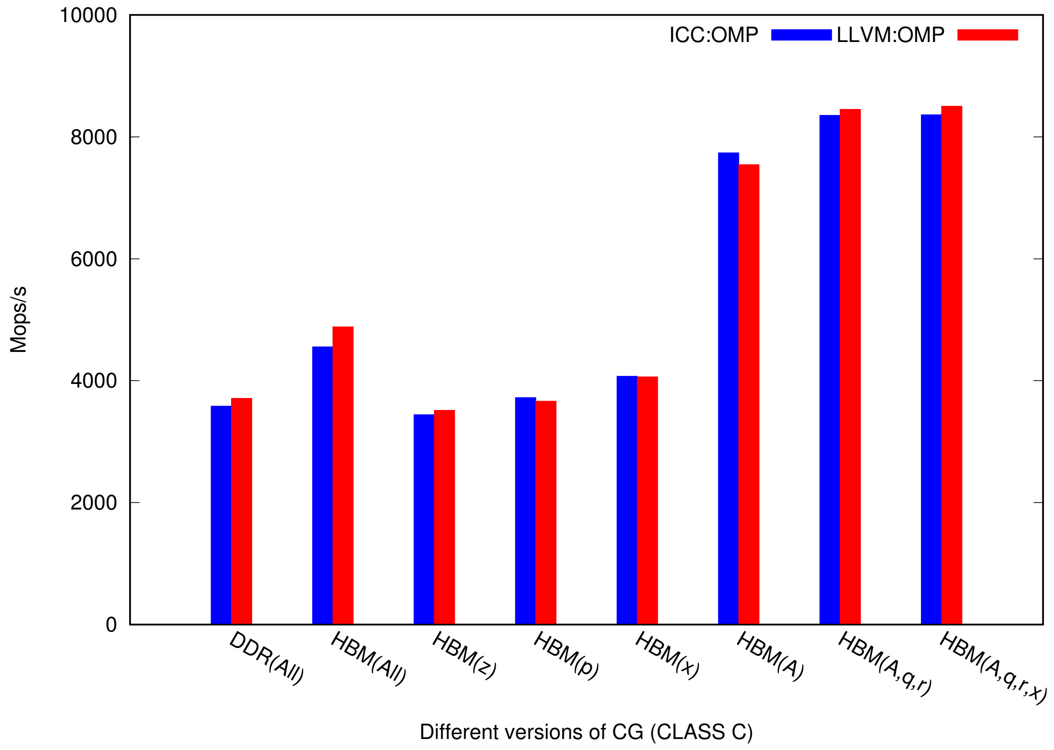
compiler-rt runtime library

Experimental Results:
Critical Data Analysis Results for the CG Benchmark

FP Array	**cost**	workshare	bandwidth	P(FP Array)
r	46	All parallel	regular	46
q	21	All parallel	regular	21
a	17	All parallel	regular	17
x	16	All parallel	regular	16
p	29	All parallel	irregular	0
Z	21	All parallel	irregular	0

Performance Results

- Setup: 1-node machine with one Intel(R) Xeon Phi(TM) CPU 7210 processor @ 1.30GHz configured with the Sub NUMA clustering mode
- LLVM 3.9, sporting Clang 3.9
- Results using:
  - Conjugate Gradient (CG) benchmark (NAS Parallel suite)
  - 2.29x performance improvement using LLVM and 2.33x using ICC



DDR vs. HBM-array-allocation performance of the OpenMP version of CG

Conclusion and Future Work

- HBM management from a compiler point-of-view
  - Decide when it is beneficial to allocate data in the HBM for sequential and OpenMP code
  - Case study: HBM (MCDRAM) of Knights Landing (KNL)
  - 2.29x performance improvement using LLVM compiler and 2.33x using Intel compiler compared to the DDR version of CG
- Future Work:
  - Improve the accuracy of our priority function
  - Implement more precise analyses regarding irregular accesses and instruction counts for recursive functions and nested loops
  - Use of AutoHBW to add size as an additional metric

Towards Automatic HBM Allocation using LLVM: A Case Study with Knights Landing

Dounia Khaldi and Barbara Chapman
Institute for Advanced Computational Science
Stony Brook University
Stony Brook, NY

The Third Workshop on the LLVM Compiler Infrastructure in HPC
Salt Lake City, Utah, November 14, 2016