OpenMP, Unified Memory, and Prefetching

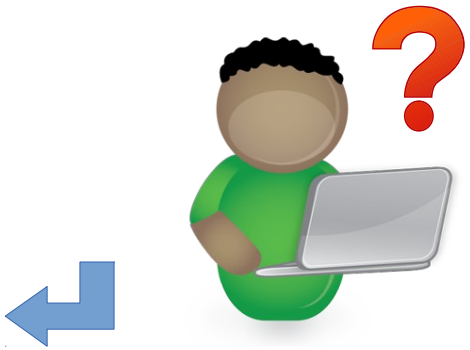
Hal Finkel[1] and Hashim Sharif[1,2]

PADAL17: 2017-08-03

[1]Argonne Leadership Computing Facility, Argonne National Laboratory

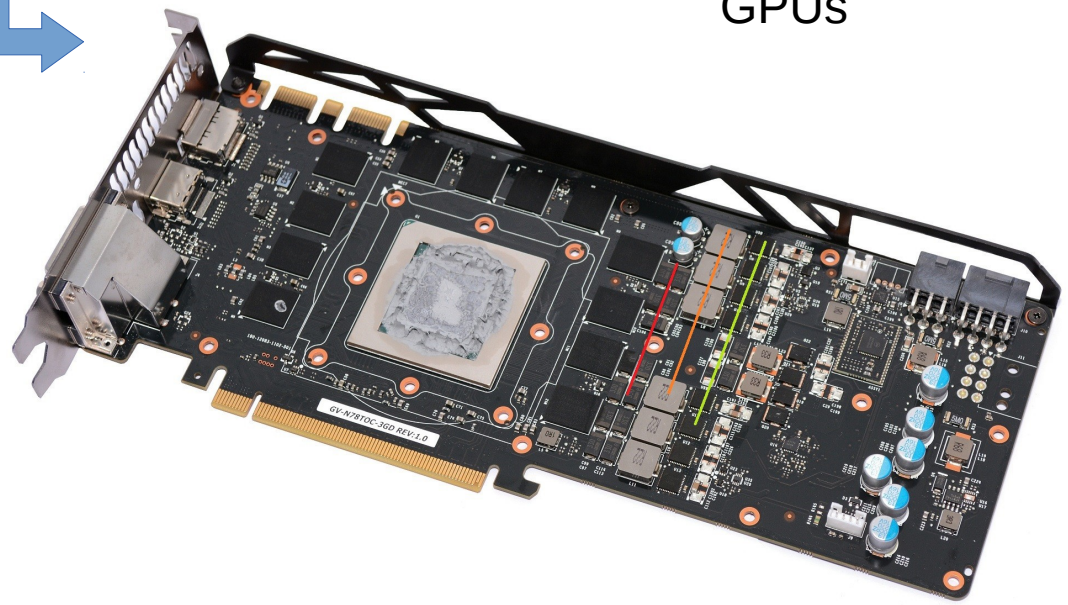
[2]Department of Computer Science, UIUC

Supercomputing “Swim Lanes”



“Many Core” CPUs

GPUs

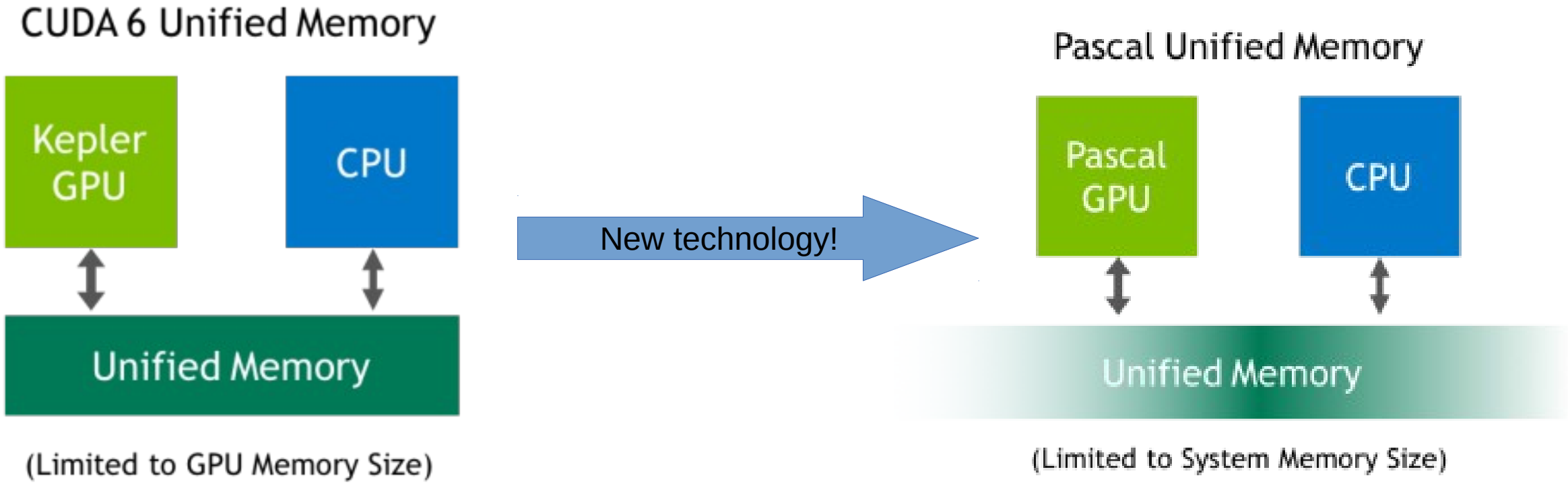


https://forum.beyond3d.com/threads/nvidia-pascal-speculation-thread.55552/page-4

http://www.nextplatform.com/2015/11/30/inside-future-knights-landing-xeon-phi-systems/

(Both will have unified memory spaces...)

CUDA Unified Memory



Unified memory enables “lazy” transfer on demand – will mitigate/eliminate the “deep copy” problem!

CUDA UM (The Old Way)

CPU Code

```
void sortfile(FILE *fp, int N) {
  char *data;
  data = (char *)malloc(N);

  fread(data, 1, N, fp);

  qsort(data, N, 1, compare);


  use_data(data);

  free(data);
}
```

CUDA 6 Code with Unified Memory

```
void sortfile(FILE *fp, int N) {
  char *data;
  cudaMallocManaged(&data, N);

  fread(data, 1, N, fp);

  qsort<<<...>>>(data,N,1,compare);
  cudaDeviceSynchronize();

  use_data(data);

  cudaFree(data);
}
```

CUDA UM (The New Way)

CPU Code

```
void sortfile(FILE *fp, int N) {
  char *data;
  data = (char *)malloc(N);

  fread(data, 1, N, fp);

  qsort(data, N, 1, compare);


  use_data(data);

  free(data);
}
```

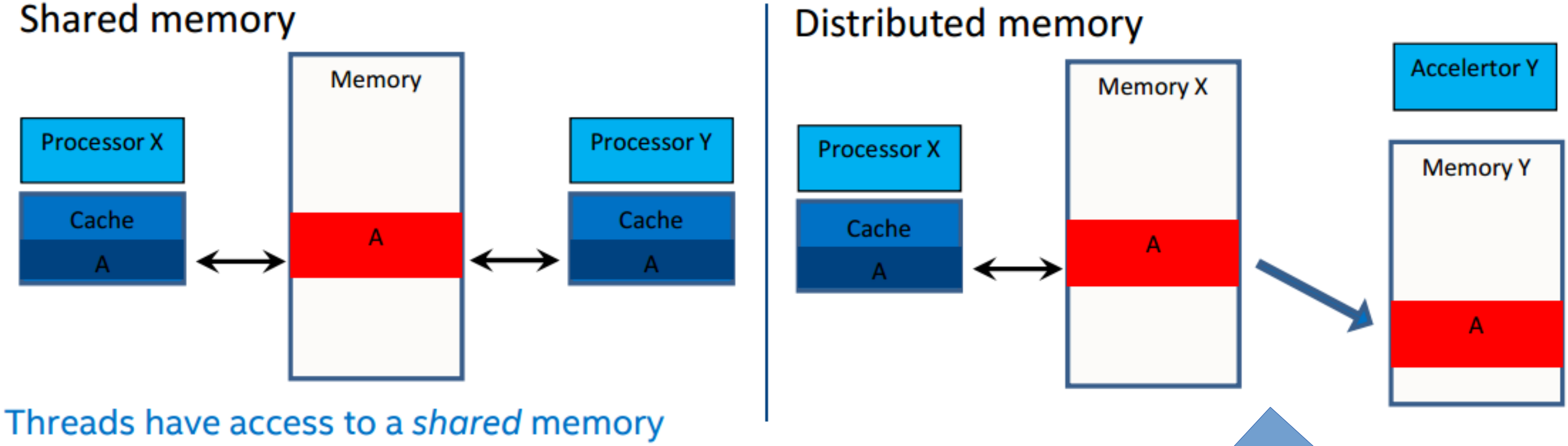
Pascal Unified Memory*

```
void sortfile(FILE *fp, int N) {
  char *data;
  data = (char *)malloc(N);

  fread(data, 1, N, fp);

  qsort<<<...>>>(data,N,1,compare);
  cudaDeviceSynchronize();

  use_data(data);

  free(data);
}
```

*with operating system support

Pointers are “the same” everywhere!

OpenMP Evolving Toward Accelerators



http://llvm-hpc2-workshop.github.io/slides/Tian.pdf

New in OpenMP 4

- To support distributed-memory systems the user is asked to provide extra information about memory dependencies.
- When we have unified memory how should we use this extra information?

OpenMP Accelerator Support – An Example (SAXPY)

```
int main(int argc, const char* argv[]) {
  float *x = (float*) malloc(n * sizeof(float));
  float *y = (float*) malloc(n * sizeof(float));
  // Define scalars n, a, b & initialize x, y









  for (int i = 0; i < n; ++i){
        y[i] = a*x[i] + y[i];
  }

  free(x); free(y); return 0;
}
```

http://llvm-hpc2-workshop.github.io/slides/Wong.pdf

OpenMP Accelerator Support – An Example (SAXPY)

```
int main(int argc, const char* argv[]) {
  float *x = (float*) malloc(n * sizeof(float));
  float *y = (float*) malloc(n * sizeof(float));
  // Define scalars n, a, b & initialize x, y

#pragma omp target data map(to:x[0:n])
{
#pragma omp target map(tofrom:y)
#pragma omp teams num_teams(num_blocks) num_threads(bsize)
```


```
#pragma omp distribute
  for (int i = 0; i < n; i += num_blocks){
```


```
#pragma omp parallel for
    for (int j = i; j < i + num_blocks; j++) {
```


```
      y[j] = a*x[j] + y[j];
  } }
} free(x); free(y); return 0; }
```

Memory transfer if necessary.

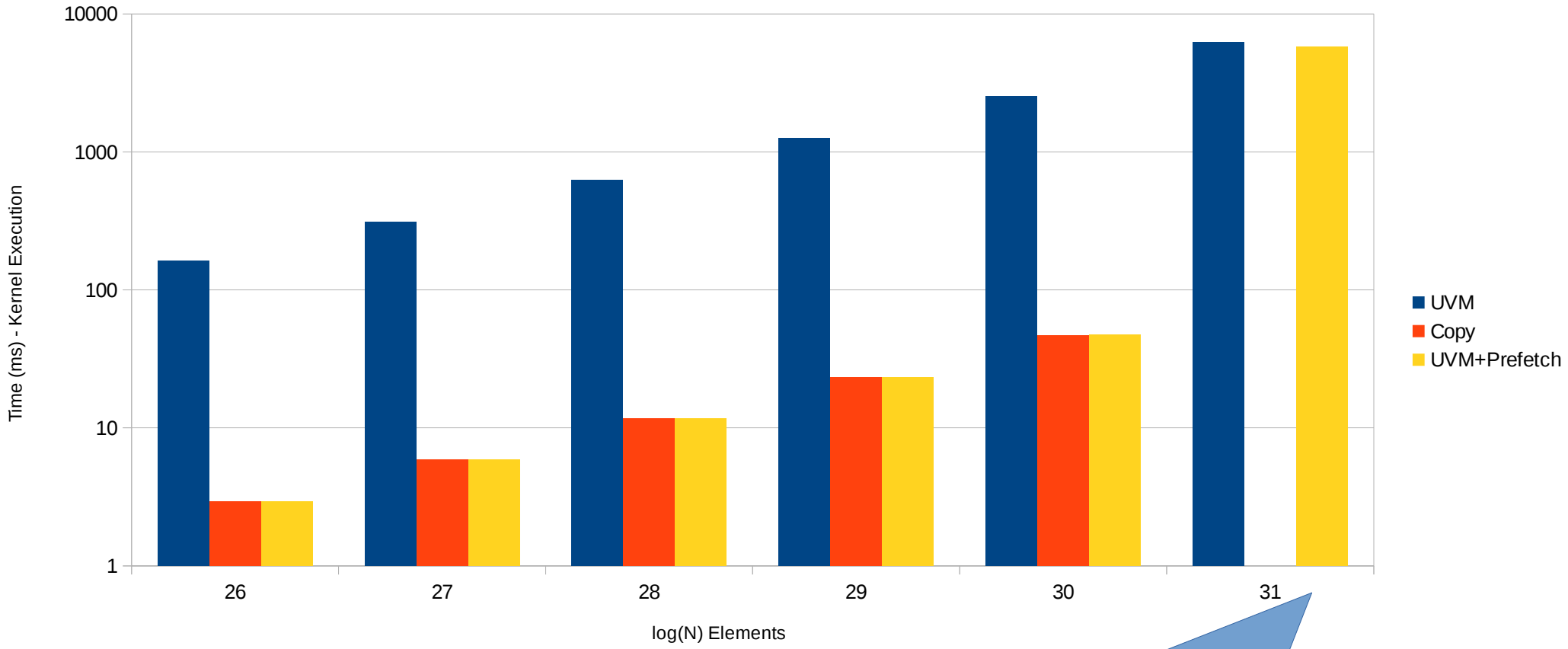
Traditional CPU-targeted OpenMP might only need this directive!

http://llvm-hpc2-workshop.github.io/slides/Wong.pdf

How To Use OpenMP Mapping Information Under UVM

- Do Nothing (i.e., allow the system to use on-demand paging).
- Ignore UVM (i.e., copy data to the device as if UVM were not there).
- Request data be moved, or “prefetched”, to the device (before kernel execution) and back to the host (after kernel execution).
- Don't move the data, but ensure that page tables and other metadata are setup for data that might be accessed on the device.

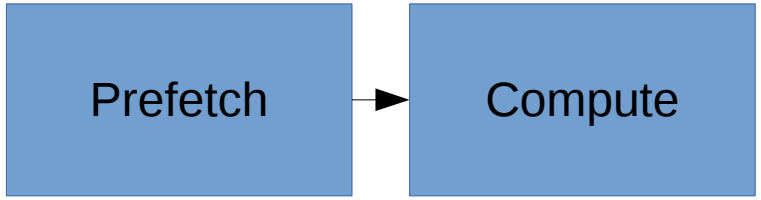
Some Performance Data - SAXPY



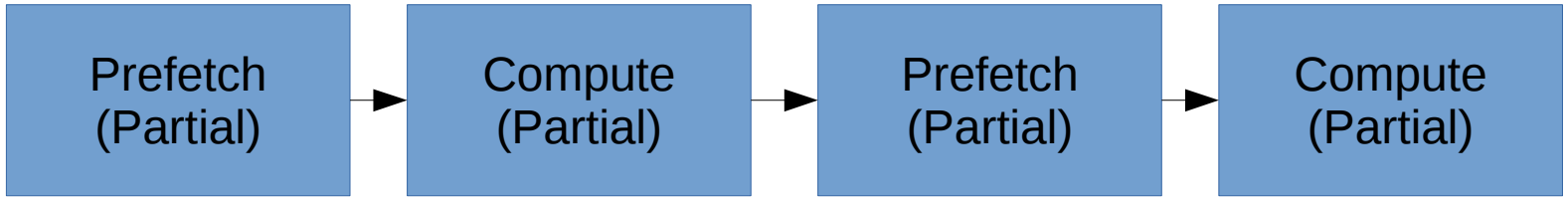
This does not fit in device memory and over-prefetching is bad!

Some Performance Data - SAXPY

This won't work if the data won't all fit...

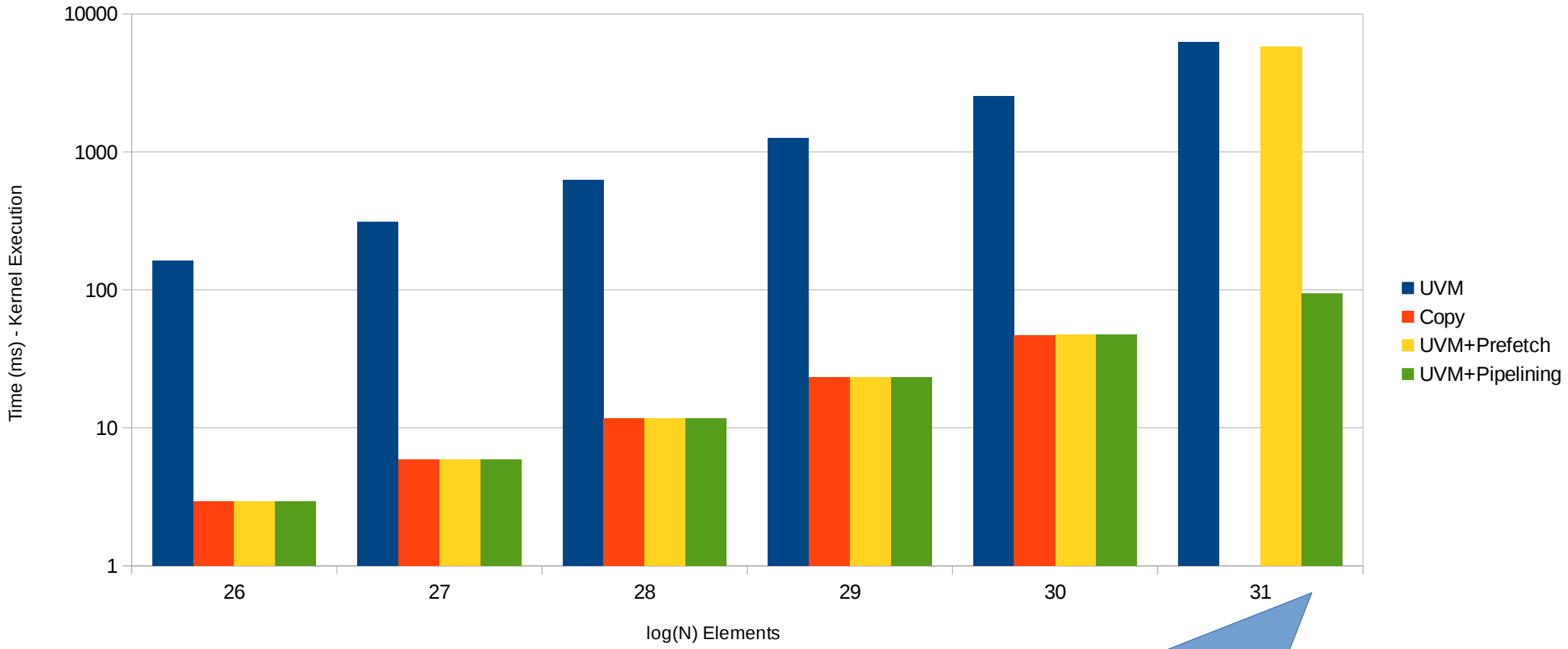


How about this (pipelining)...



(Now everything fits)

Some Performance Data - SAXPY



Good performance with pipelining!

When On-Demand Prefetching Is Good...

```
for (int I = 0; I < N; ++I) {
  something(A[I], B[I], C[I]);
  if (test(A[I], B[I], C[I])) {
    something_else(D[I], E[I]);
  }
}
```

If this is rarely true...

Then on-demand fetching of data from D and E might be cheaper than the cost of preemptive copying!

$Cost_{preemptive}$ = (cost per page for prefetch) * (total size)/(page size)

$Cost_{on\text{-}demand}$ = (cost per page for on-demand fetch) * (total size)/(page size) * (probability of access)

When On-Demand Prefetching Is Good...

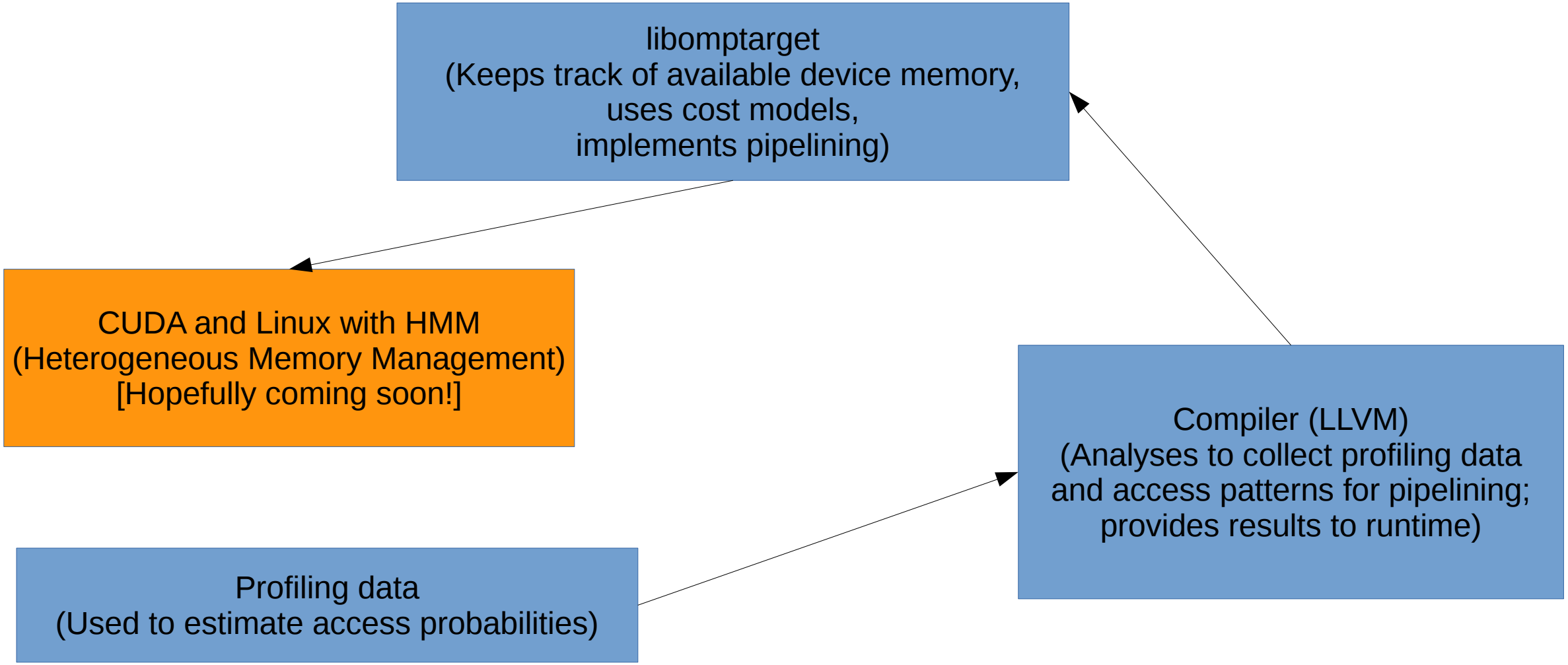
p = probability of access.

$(Cost_{on\text{-}demand} / Cost_{preemptive})(p) = 1$ for p = 1/3 based on testing on OLCF's SummitDev (P100/NVLINK)

If the access probability is, on average, greater than this (per page) then it is better to prefetch (if everything will fit).

How much space is available on the device? With UVM, only the OpenMP runtime can estimate this...

The Big Picture



Acknowledgments

- The LLVM community (including our many contributing vendors)
- The SOLLVE project (part of the Exascale Computing Project)
- Other members of the SOLLVE project (Lingda Li, Barbara Chapman, Bronis de Supinski) and Vikram Adve
- OLCF, ORNL for access to the SummitDev system.
- ALCF, ANL, and DOE
- ALCF is supported by DOE/SC under contract DE-AC02-06CH11357

The Fourth Workshop on the LLVM Compiler Infrastructure in HPC

Workshop held in conjunction with SC17 – Monday, November 13, 2017 – Denver, Colorado, USA

View On GitHub

Topics of interest include, but are not limited to:

- Compiler design for highly-concurrent/parallel environments
- Compilation techniques targeted at high-performance computing codes
- Programming-language implementation techniques enabling high performance and high productivity
- Embedding compilation and dynamic execution at scale
- Tools for optimization, profiling, and feedback
- Source code transformation and analysis
- Gap analyses of open-source LLVM-based tools

NEW THIS YEAR: The workshop will hold a lightning-talk session. Please contribute to making this session both vibrant and informative! An abstract and one-page summary is required for consideration.

Deadlines

- Paper submissions due: September 1, 2017

https://llvm-hpc4-workshop.github.io/