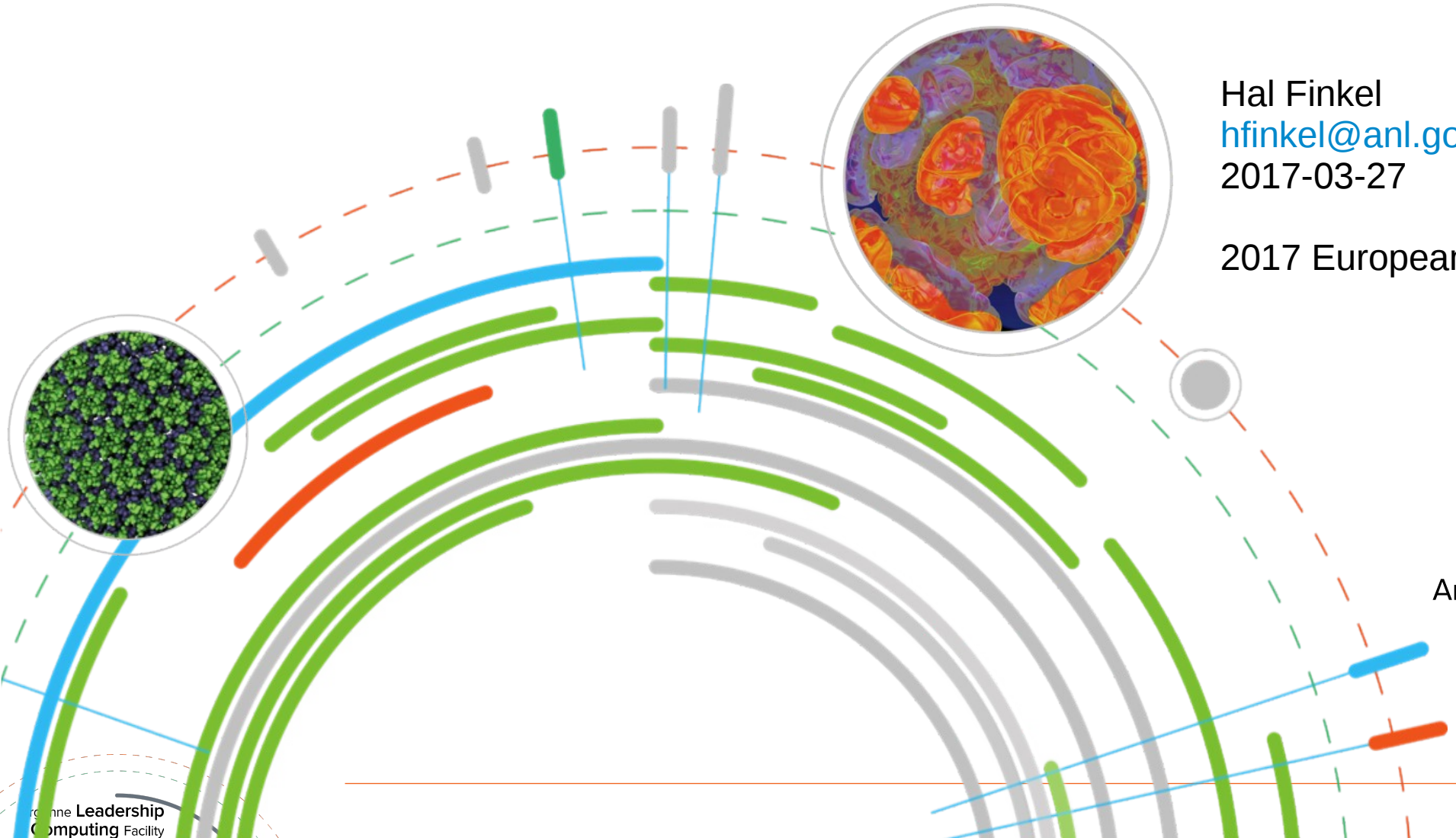
# LLVM for the future of Supercomputing



Hal Finkel
hfinkel@anl.gov
2017-03-27

2017 European LLVM Developers' Meeting

Argonne **Leadership Computing** Facility

Argonne NATIONAL LABORATORY

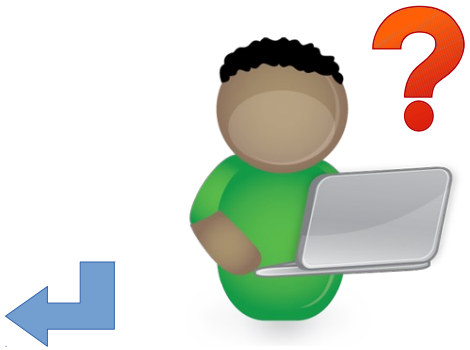
# What is Supercomputing?

Computing for large, tightly-coupled problems.

Lots of computational
capacity
lots of high

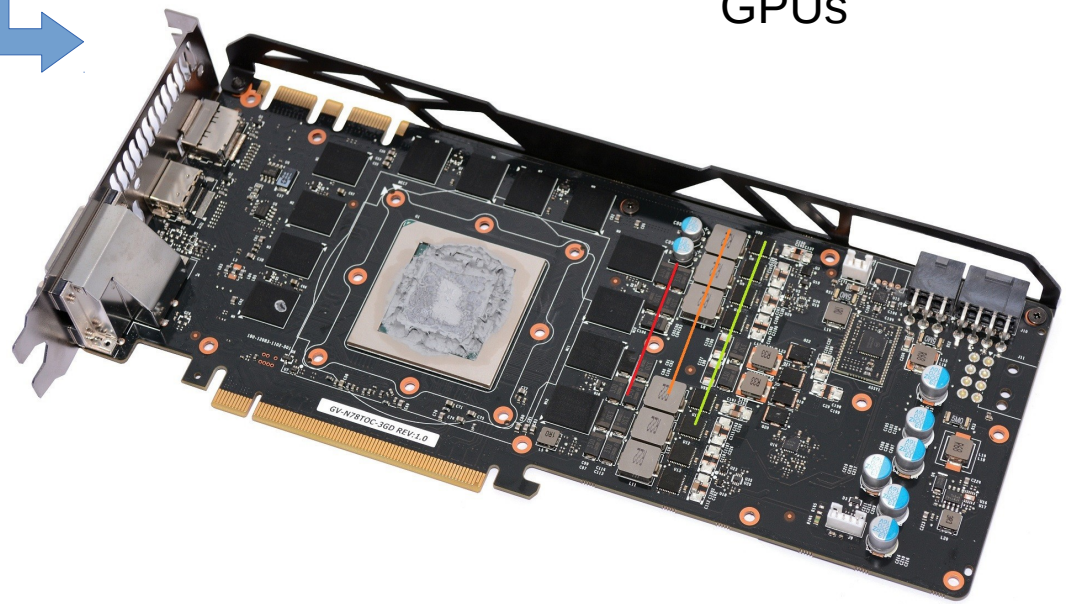
High computational density paired with a high-throughput low-latency network.

# 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/

| System attributes | NERSC Now | OLCF Now | ALCF Now | NERSC Upgrade | OLCF Upgrade | ALCF Upgrades | |
|---|---|---|---|---|---|---|---|
| Name<br>Planned Installation | **Edison** | **TITAN** | **MIRA** | **Cori<br>2016** | **Summit<br>2017-2018** | **Theta<br>2016** | **Aurora<br>2018-2019** |
| System peak (PF) | 2.6 | 27 | 10 | > 30 | **200** | >8.5 | 180 |
| Peak Power (MW) | 2 | 9 | 4.8 | | | | 13 |
| Total system memory | 357 TB | 710TB | 768TB | ~1 PB DDR<br>High Bandwidth<br>M | | 0 TB DDR4 + | > 7 PB High Bandwidth On- |
| Node performance (TF) | 0.460 | 1 | | | | | |
| Node processors | Intel Ivy Bridge | A<br>Op<br>Nvidia Kepler | PowerPC A2 | core CPUs<br>Intel Haswell CPU in data partition | &<br>**multiple Nvidia Voltas GPUS** | Xeon Phi many core CPUs | Knights Hill Xeon Phi many core CPUs |
| | 5,600 | 18,688 | 49,152 | 9,300 nodes<br>data partition | | | >50,000 nodes |
| | | | 5D Torus | Aries | **Dual Rail EDR-IB** | Aries | 2nd Generation Intel Omni-Path Architecture |
| File System | 168 GB/s, Lustre® | 1 TB/s, Lustre® | 26 PB<br>300 GB/s<br>GPFS™ | 28 PB<br>744 GB/s<br>Lustre® | **120 PB<br>1 TB/s<br>GPFS™** | 10PB, 210 GB/s<br>Lustre initial | 150 PB<br>1 TB/s<br>Lustre® |

Only a 2.7x increase in power!

Our next system will have:
180 PF
An 18x increase!

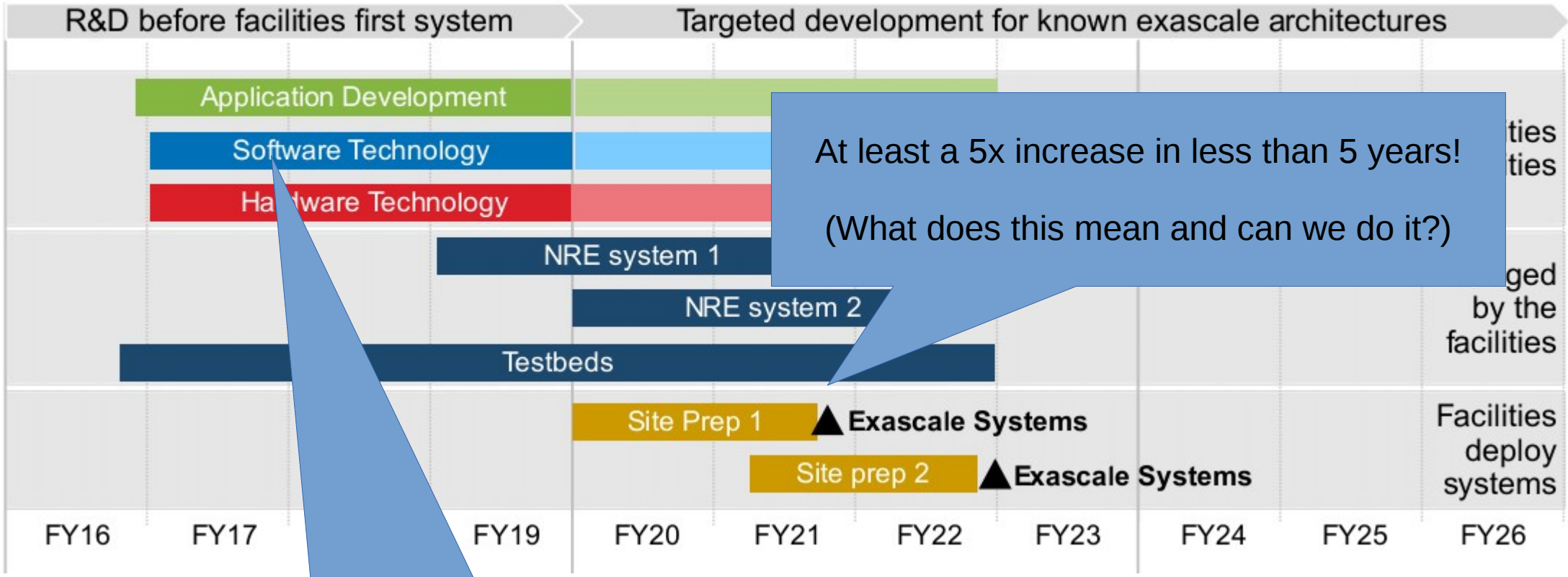
n system is:
OPS)

Still ~50,000 nodes.

The heterogeneous system with GPUs has 10x fewer nodes!

http://science.energy.gov/~/media/ascr/ascac/pdf/meetings/201604/2016-0404-ascac-01.pdf

# High-level ECP technical project schedule




At least a 5x increase in less than 5 years!

(What does this mean and can we do it?)

We need to start preparing applications and tools now.

scaleproject.org for more information.

https://exascaleproject.org/wp-content/uploads/2017/03/Messina_ECP-IC-Mar2017-compressed.pdf

# What Exascale Means To Us...

| Exascale System | Goal |
|---|---|
| Delivery Date | 2019-2020 |
| Performance | 1000 PF LINPACK and 300 PF on to-be-specified applications |
| Power Consumption* | 20 MW |
| MTBAI** | 6 days |
| Memory including NVRAM | 128 PB |
| Node Memory Bandwidth | 4 TB/s |
| Node Interconnect Bandwidth | 400 GB/s |

*Power consumption includes only power to the c
storage or cooling systems.
**The mean time to application failure requiring a
must be greater than 24 hours, and the asymptotic
over time. The system overhead to handle automa
application efficiency by more than half.

http://estrfi.cels.anl.gov/files/2011/07/RF

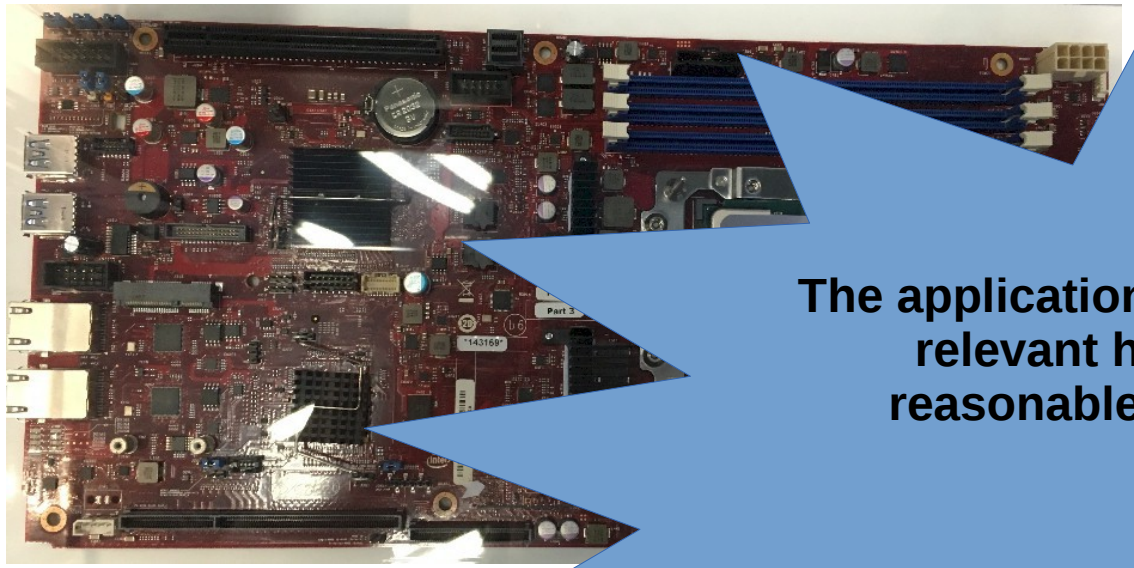
It means 5x the compute and 20x the memory on 1.5x the power!

# What do we want?

# We Want Performance Portability!

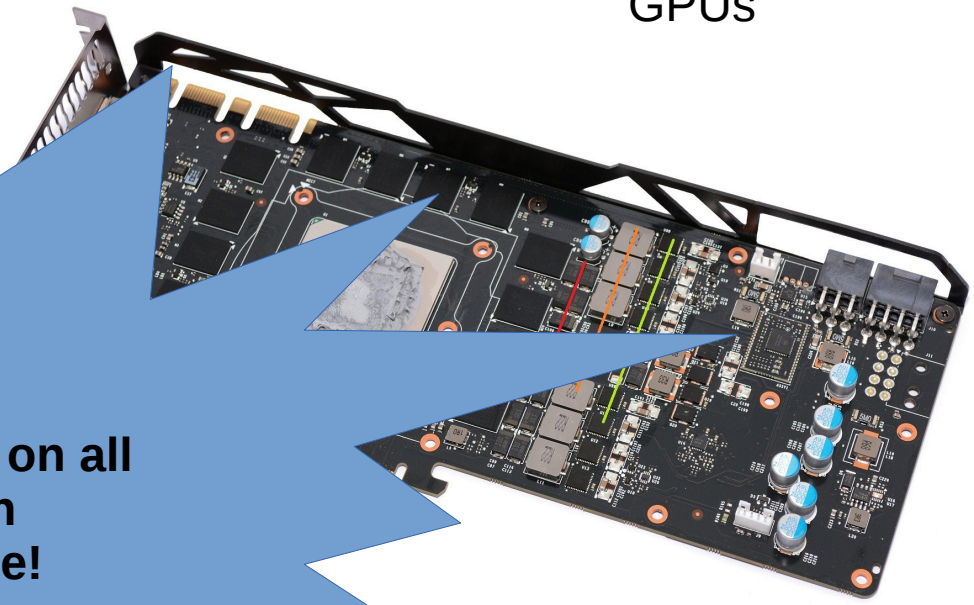
Application
(One Maintainable Code Base)

“Many Core” CPUs



http://www.nextplatform.com/2015/1

GPUs



m/threads/nv cal-speculation-thread.55552/page-4

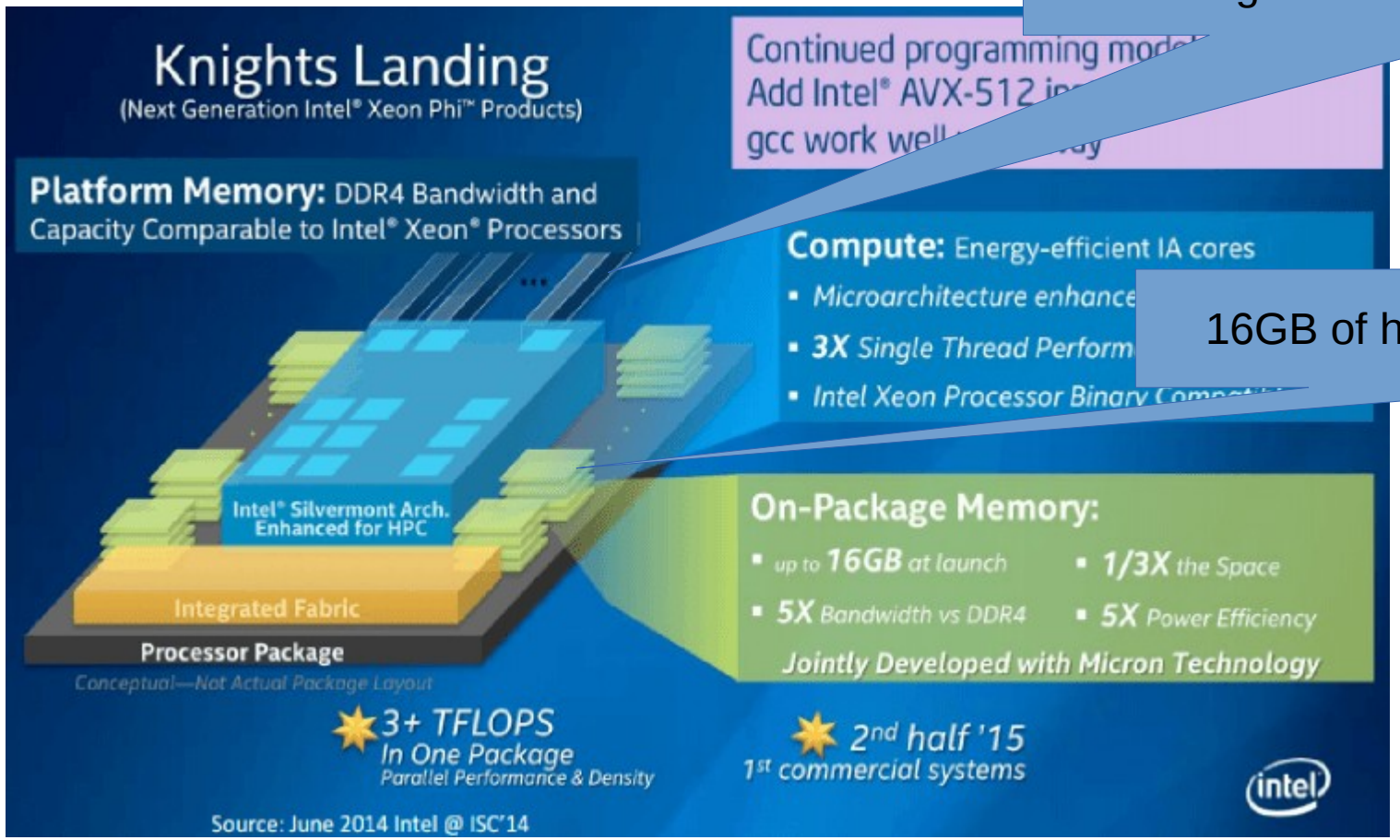
**The application should run on all relevant hardware with reasonable performance!**

# Let's Talk About Memory...

# Intel Xeon Phi HBM

Large amounts of regular DRAM far away.

16GB of high-bandwidth on-package memory!

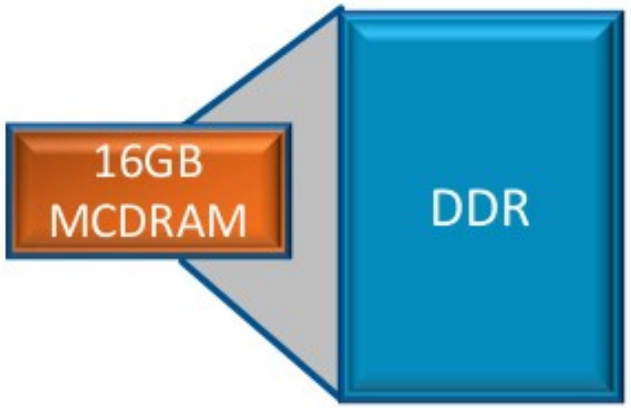



http://www.techenablement.com/preparing-knights-landing-stay-hbm-memory/

# Intel Xeon Phi HBM Modes

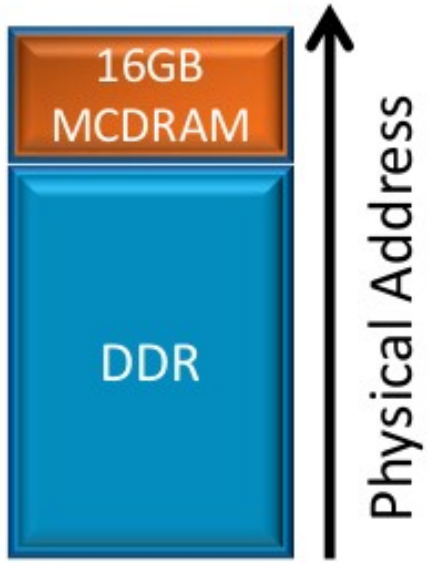
**Three** Modes. Selected at boot

## Cache Mode



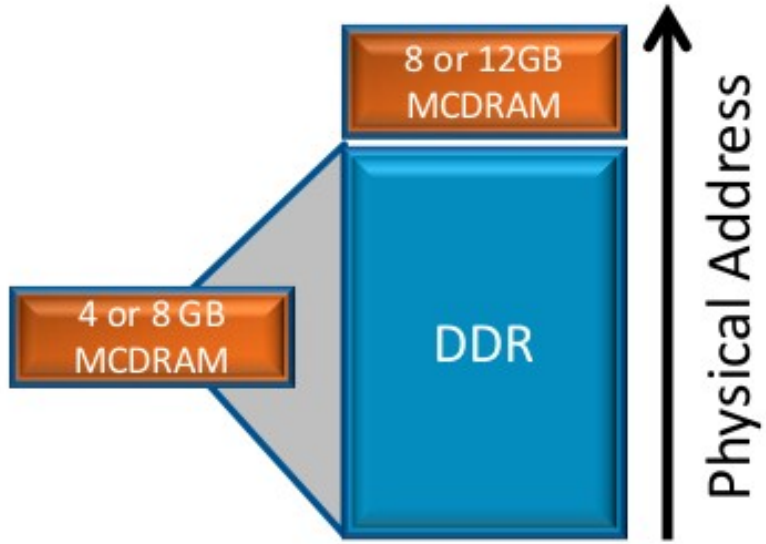

- SW-Transparent, Mem-side cache
- Direct mapped. 64B lines.
- Tags part of line
- Covers whole DDR range

## Flat Mode



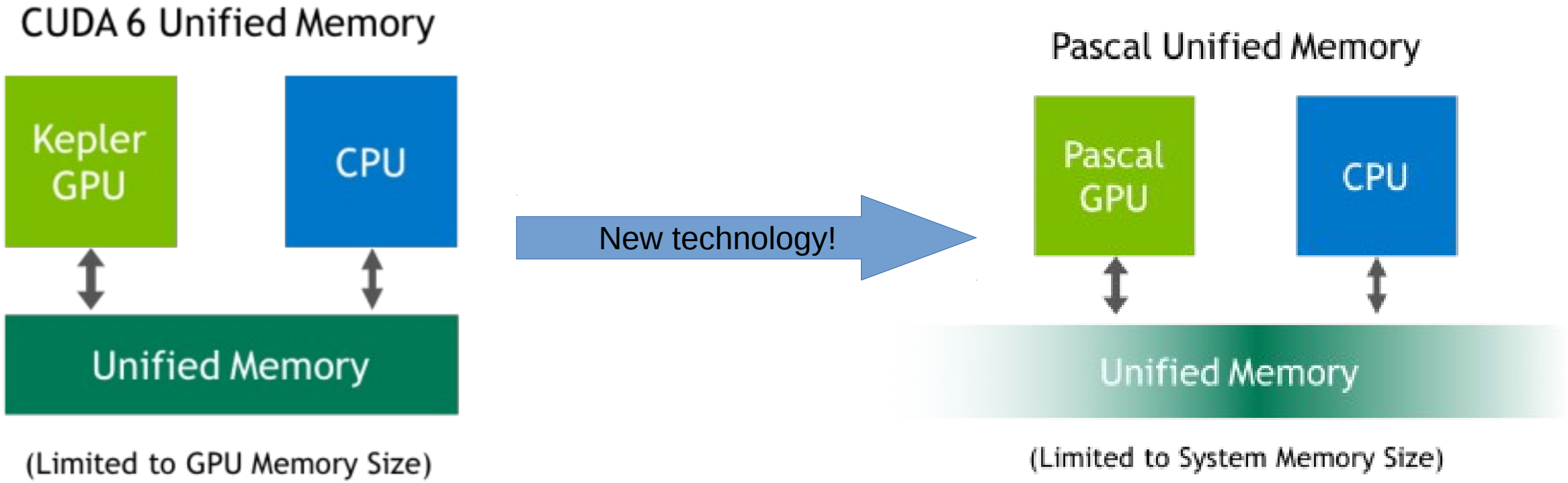

- MCDRAM as regular memory
- SW-Managed
- Same address space

## Hybrid Mode




- Part cache, Part memory
- 25% or 50% cache
- Benefits of both

# 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!

# How Do We Get Performance Portability?

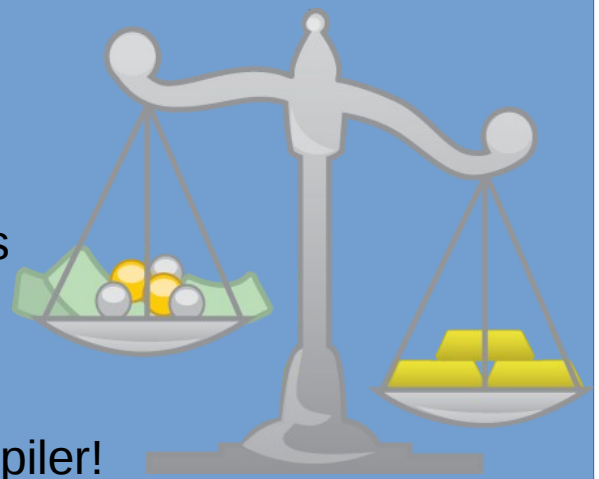
# How Do We Get Performance Portability? Shared Responsibility!

Applications and Solver Libraries

Applications and solver libraries must be flexible and parameterized! Why?

Trade-offs between...

- basis functions
- resolution
- Lagrangian vs. Eulerian representations
- renormalization and regularization schemes
- solver techniques
- evolved vs computed degrees of freedom
- and more…

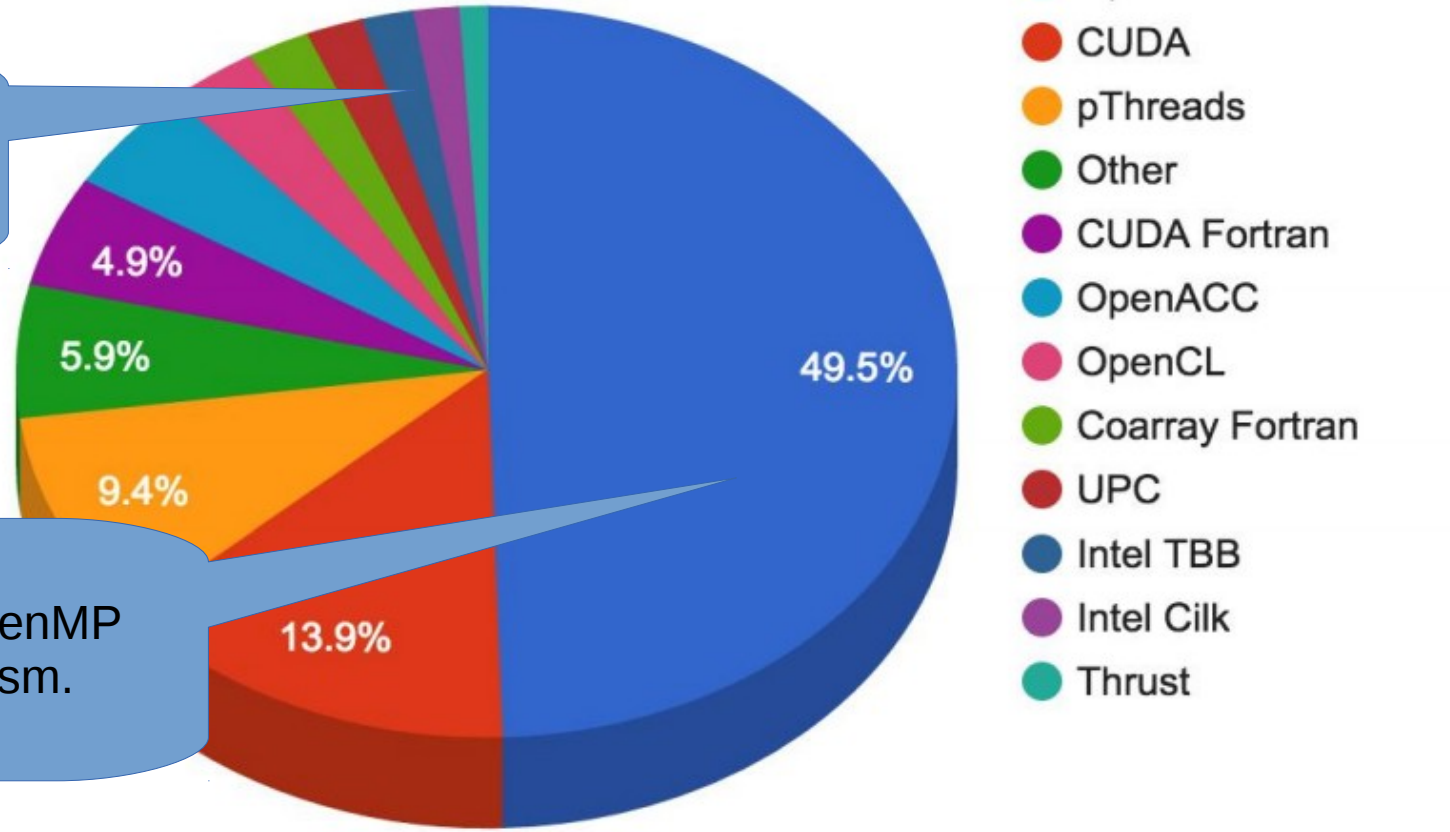


cannot be made by a compiler!

Autotuning can help.

# How do we express parallelism - MPI+X?

✓ OpenMP is about 50%, out of all choices of X




A minority of applications use abstraction libraries (TBB and Thrust on this chart)

In 2015, many codes use OpenMP directly to express parallelism.

Courtesy of Yun (Helen) He, Alice Koniges, et. al., (NERSC) at OpenMPCon'2015

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

## How do we express parallelism - MPI+X?

But this is changing…

- We're seeing even greater adoption of OpenMP, but…
- Many applications are not using OpenMP directly. Abstraction libraries are gaining in popularity.

Often uses OpenMP and/or other compiler directives under the hood.

Use of C++ Lambdas.

…TBB and Thrust.

- RAJA (https://github.com/LLNL/RAJA)

```
RAJA::ReduceSum<reduce_policy, double> piSum(0.0);

RAJA::forall<execute_policy>(begin, numBins, [=](int i) {
  double x = (double(i) + 0.5) / numBins;
  piSum += 4.0 / (1.0 + x * x);
});
```

- Kokkos (https://github.com/kokkos)

```
int sum = 0;
// The KOKKOS_LAMBDA macro replaces
// the capture-by-value clause [=].
// It also handles any other syntax
// needed for CUDA.
Kokkos::parallel_reduce (n, KOKKOS_LAMBDA (const int i,
                                           int& lsum) {
    lsum += i*i;
  }, sum);
```

## How do we express parallelism - MPI+X?

And starting with C++17, the standard library has parallel algorithms too...

Table 2 — Table of parallel algorithms

| | | | |
|---|---|---|---|
| adjacent_difference | adjacent_find | all_of | any_of |
| copy | copy_if | copy_n | count |
| count_if | equal | exclusive_scan | fill |
| fill_n | find | find_end | find_first_of |
| find_if | find_if_not | for_each | for_each_n |
| generate | generate_n | includes | inclusive_scan |
| inner_product | inplace_merge | is_heap | is_heap_until |
| is_partitioned | is_sorted | is_sorted_until | lexicographical_compare |
| max_element | merge | min_element | minmax_element |
| mismatch | move | none_of | nth_element |
| partial_sort | partial_sort_copy | partition | partition_copy |
| reduce | remove | remove_copy | remove_copy_if |
| remove_if | replace | replace_copy | replace_copy_if |
| replace_if | reverse | reverse_copy | rotate |
| rotate_copy | search | search_n | set_difference |
| set_intersection | set_symmetric_difference | set_union | sort |
| stable_partition | stable_sort | swap_ranges | transform |
| transform_exclusive_scan | transform_inclusive_scan | transform_reduce | uninitialized_copy |
| uninitialized_copy_n | uninitialized_fill | uninitialized_fill_n | unique |
| unique_copy | | | |

[ *Note:* Not all algorithms in the Standard Library have counterparts in Table 2. — *end note* ]

```
// For example:
std::sort(std::execution::par_unseq, vec.begin(), vec.end()); // parallel and vectorized
```

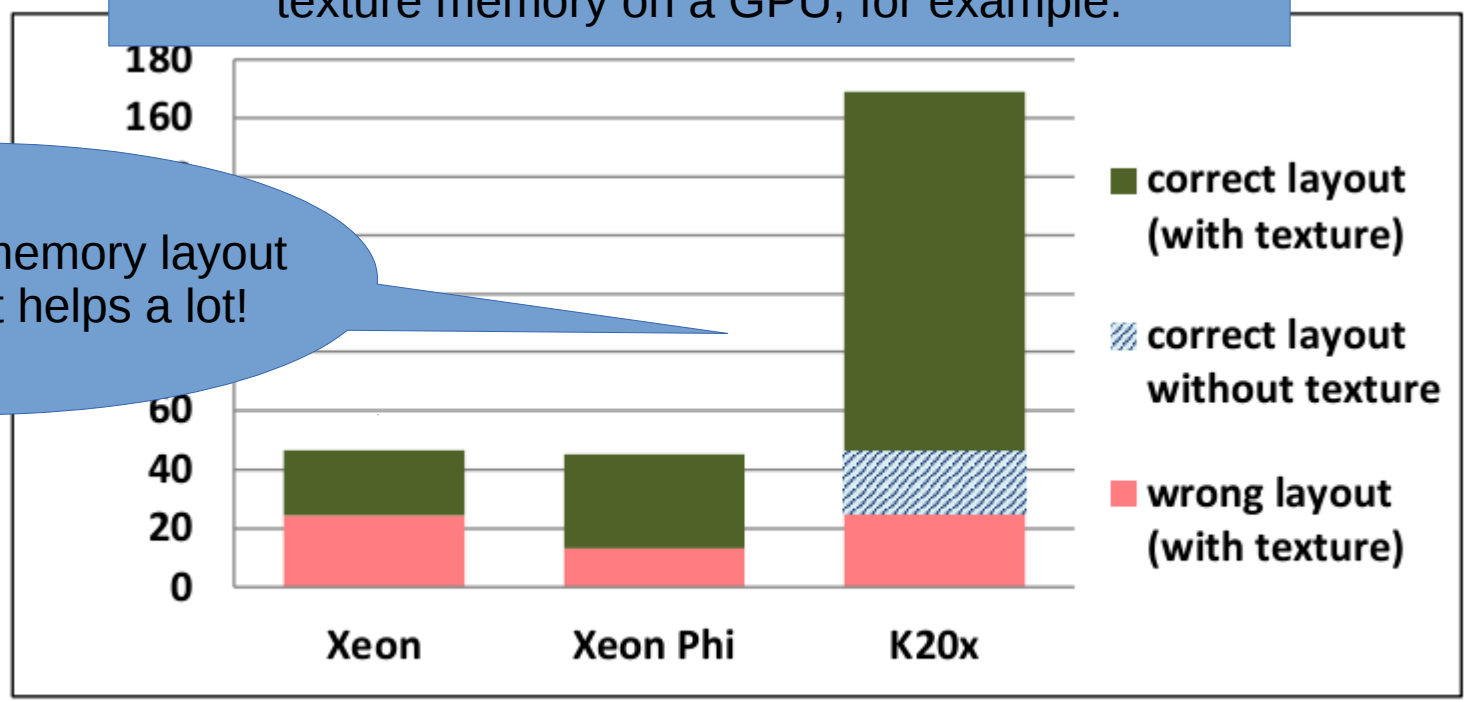
# What About Memory?

It is really hard for compilers to change memory layouts and generally determine what memory is needed where. The Kokkos C++ library has memory placement and layout policies:

View<const double ***, Layout, Space , MemoryTraits<RandomAccess>> name (...);

Constant random-access data might be put into texture memory on a GPU, for example.

Using the right memory layout and placement helps a lot!




- **Large loss in performance with wrong layout**

https://trilinos.org/oldsite/events/trilinos_user_group_2013/presentations/2013-11-TUG-Kokkos-Tutorial.pdf

# The Exascale Computing Project – Improvements at All Levels

Applications
and Solver Libraries

Over 30 Application and Library Teams

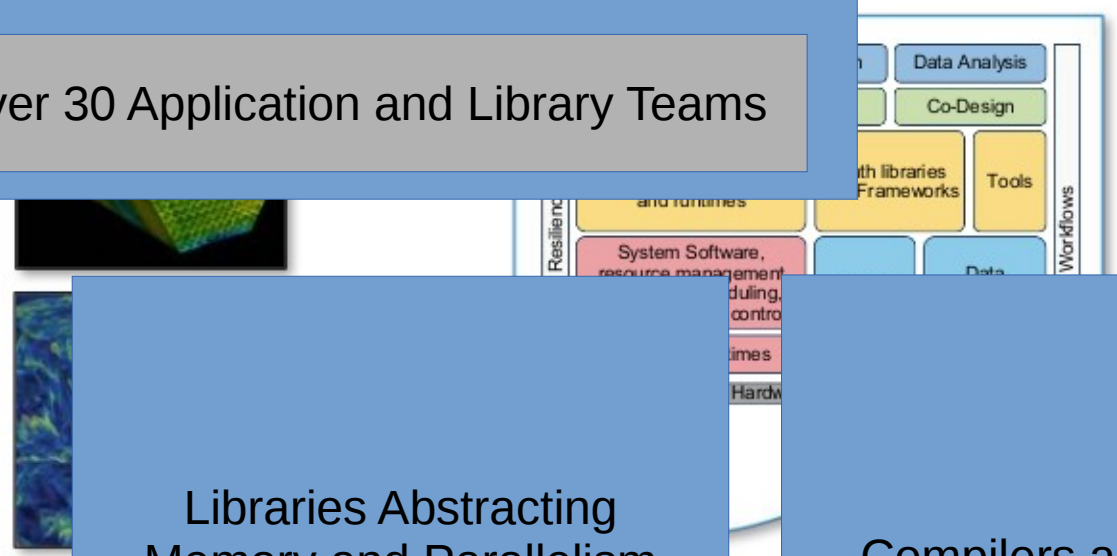
Libraries Abstracting
Memory and Parallelism

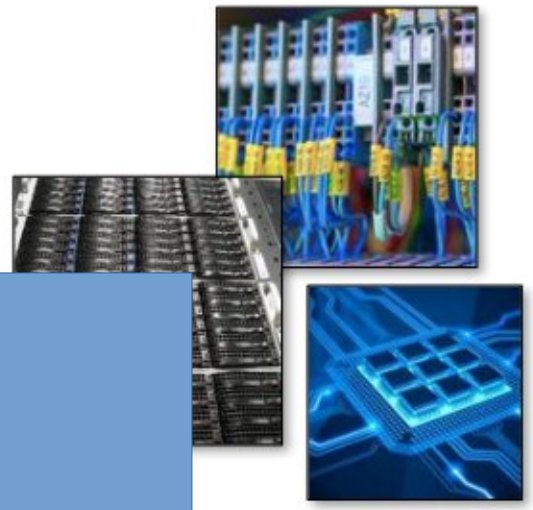
Kokkos, RAJA, etc.

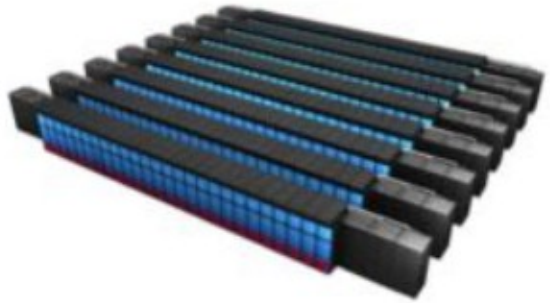
Compilers and Tools

SOLLVE, PROTEAS,
Y-Tune, ROSE,
Flang, etc.

| re ogy | Hardware Technology | Exascale Systems |
| --- | --- | --- |
| e and software k | Hardware technology elements | Integrated exascale supercomputers |



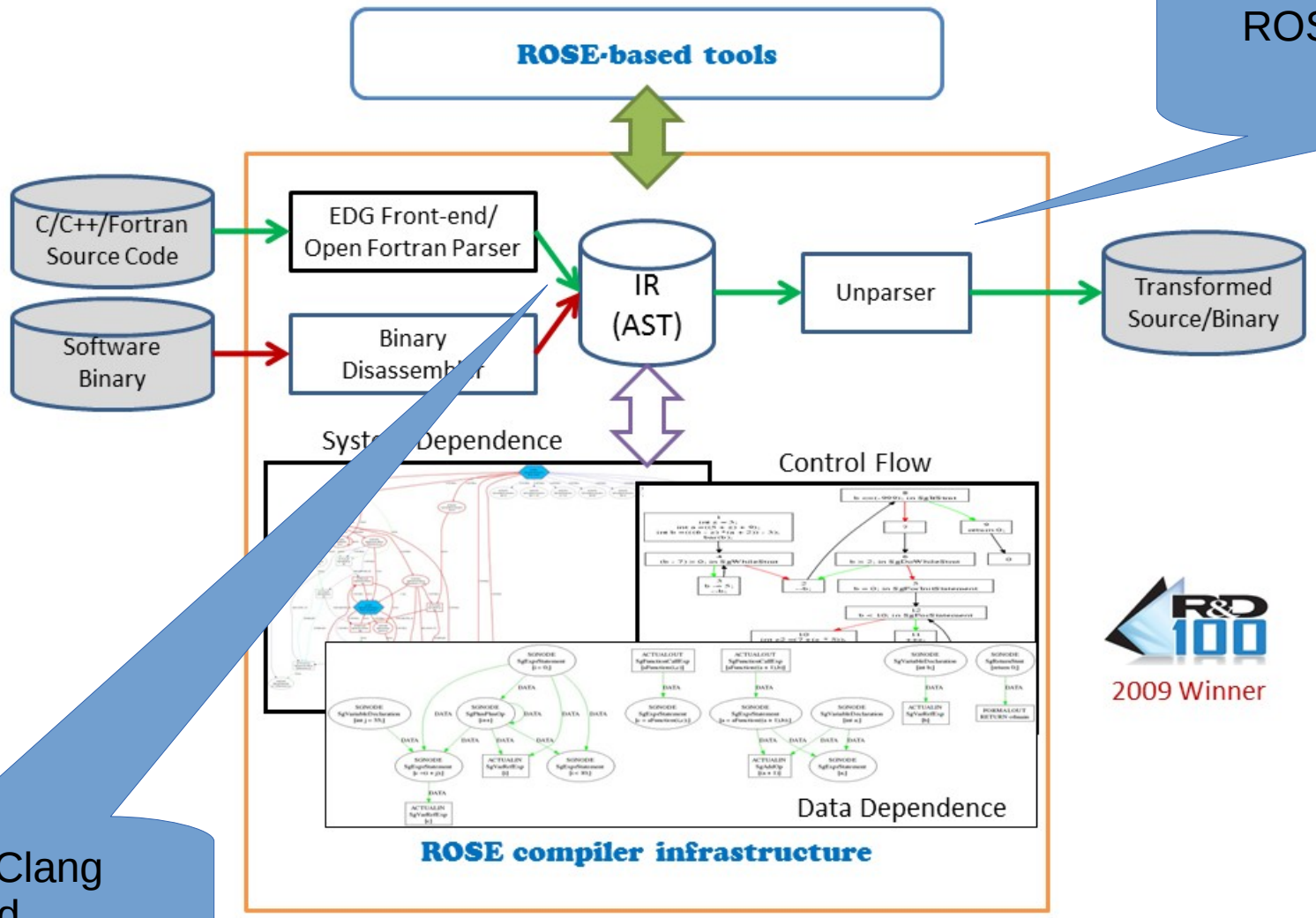


EXASCALE COMPUTING PROJECT

# Now Let's Talk About LLVM...

# LLVM Development in ECP

# ROSE – Advanced Source-to-Source Rewriting




ROSE can generate LLVM IR.

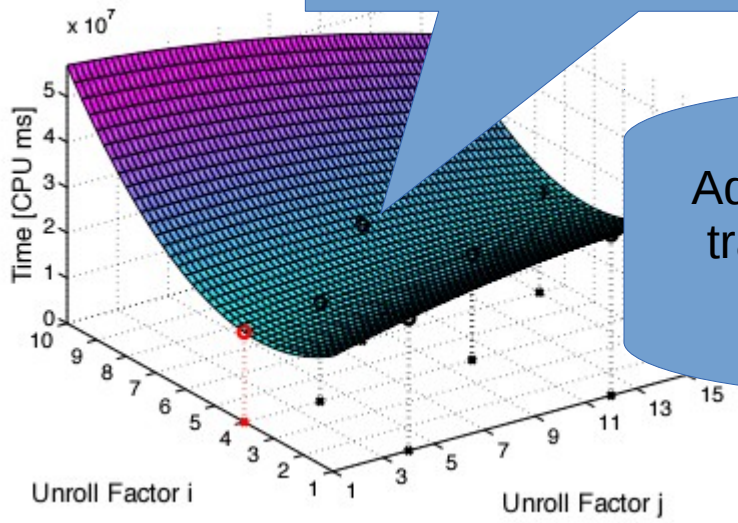
ROSE can use Clang as a frontend.

R&D 100 2009 Winner

http://rosecompiler.org/

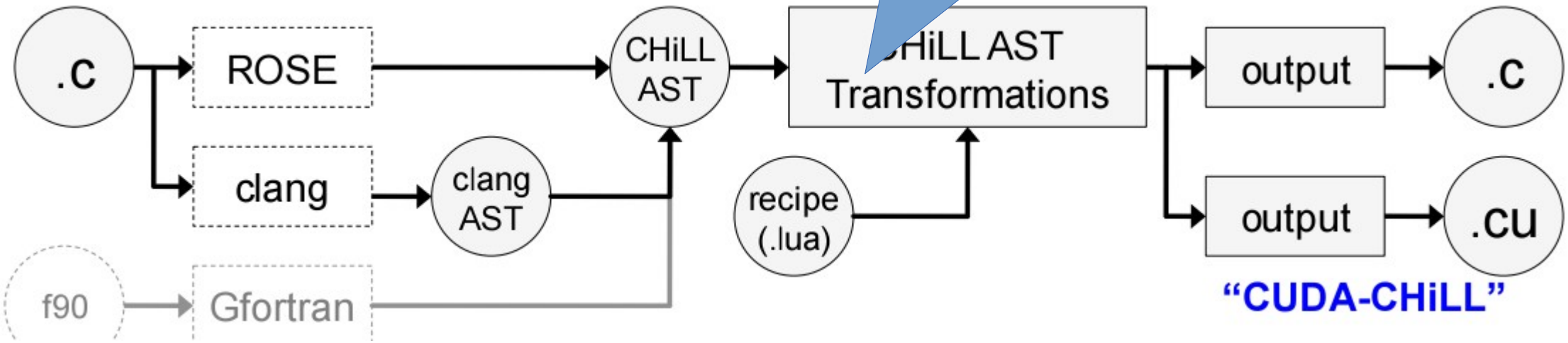
# Y-Tune

Machine-learning assisted search and optimization.

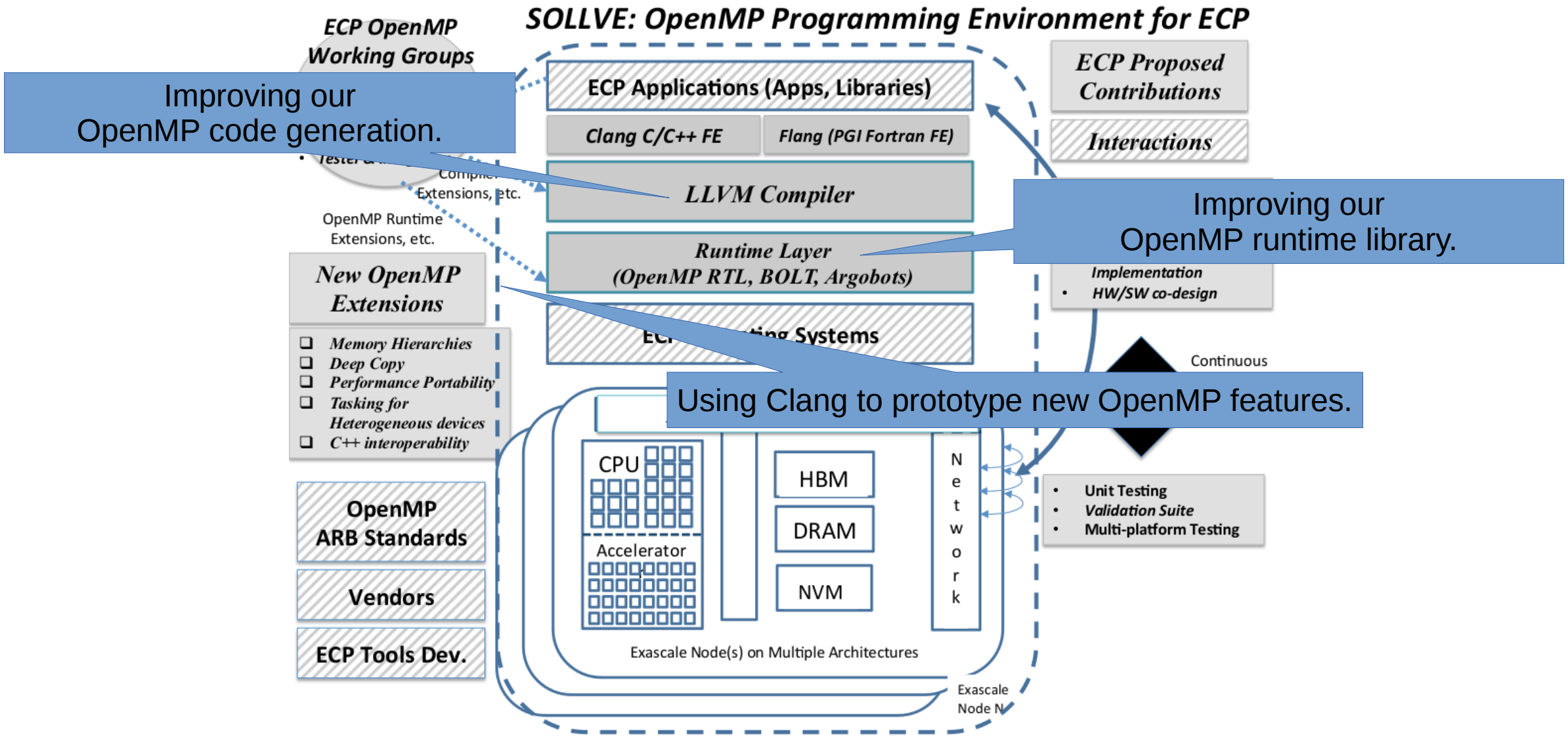



Y-Tune's scope includes improving LLVM for:
- Better optimizer feedback to guide search
  (e.g. via pragmas)

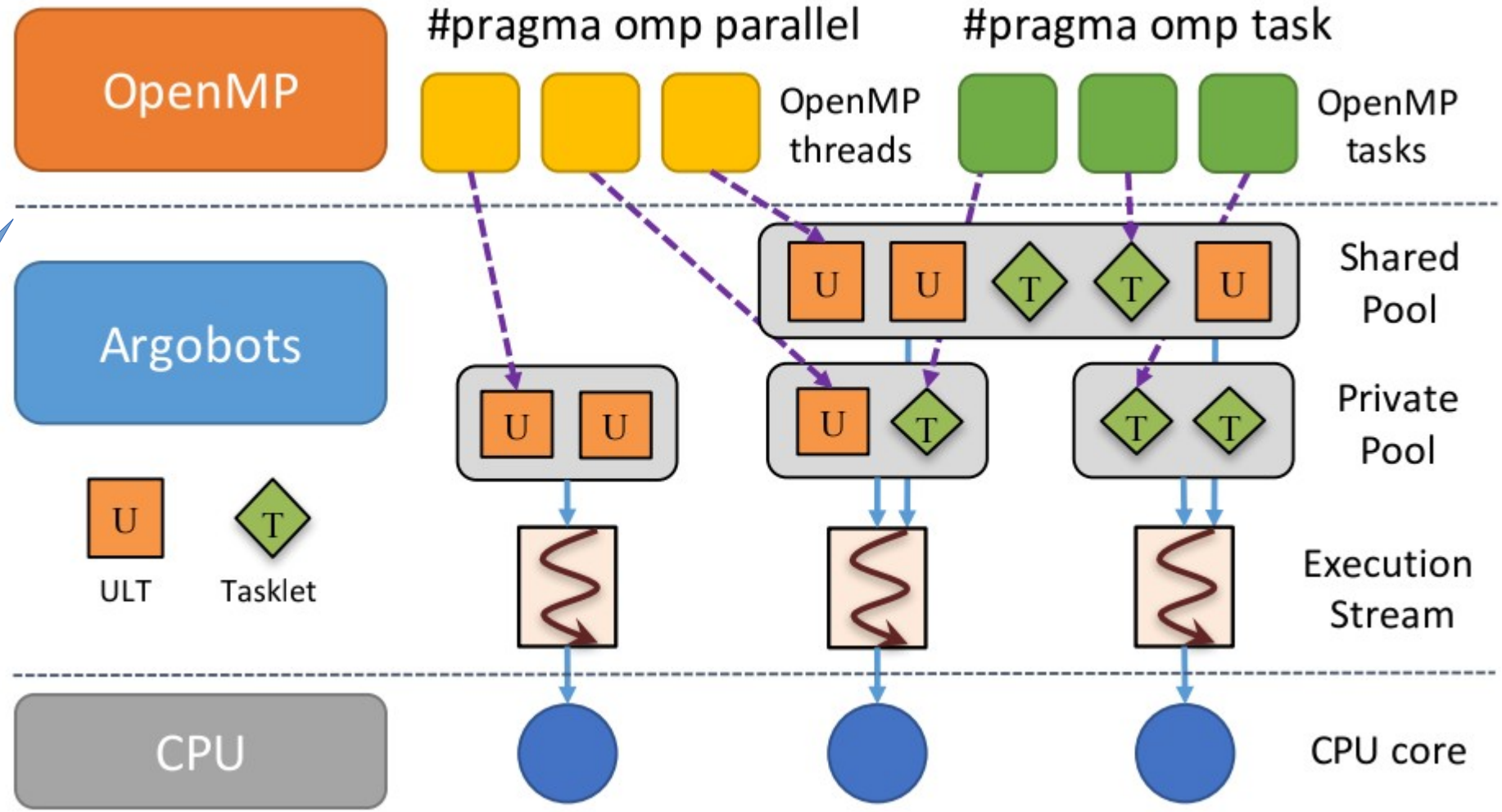
Advanced polyhedral and **application-specific** operator transformations. We can deal with the combined space of compiler-assisted and algorithm tuning!



# SOLLVE – "**S**caling **O**penmp with **LLV**m for **E**xascale performance and portability"



# BOLT - “**B**OLT is **O**penMP over **L**ightweight **T**hreads” (Now Part of SOLLVE)



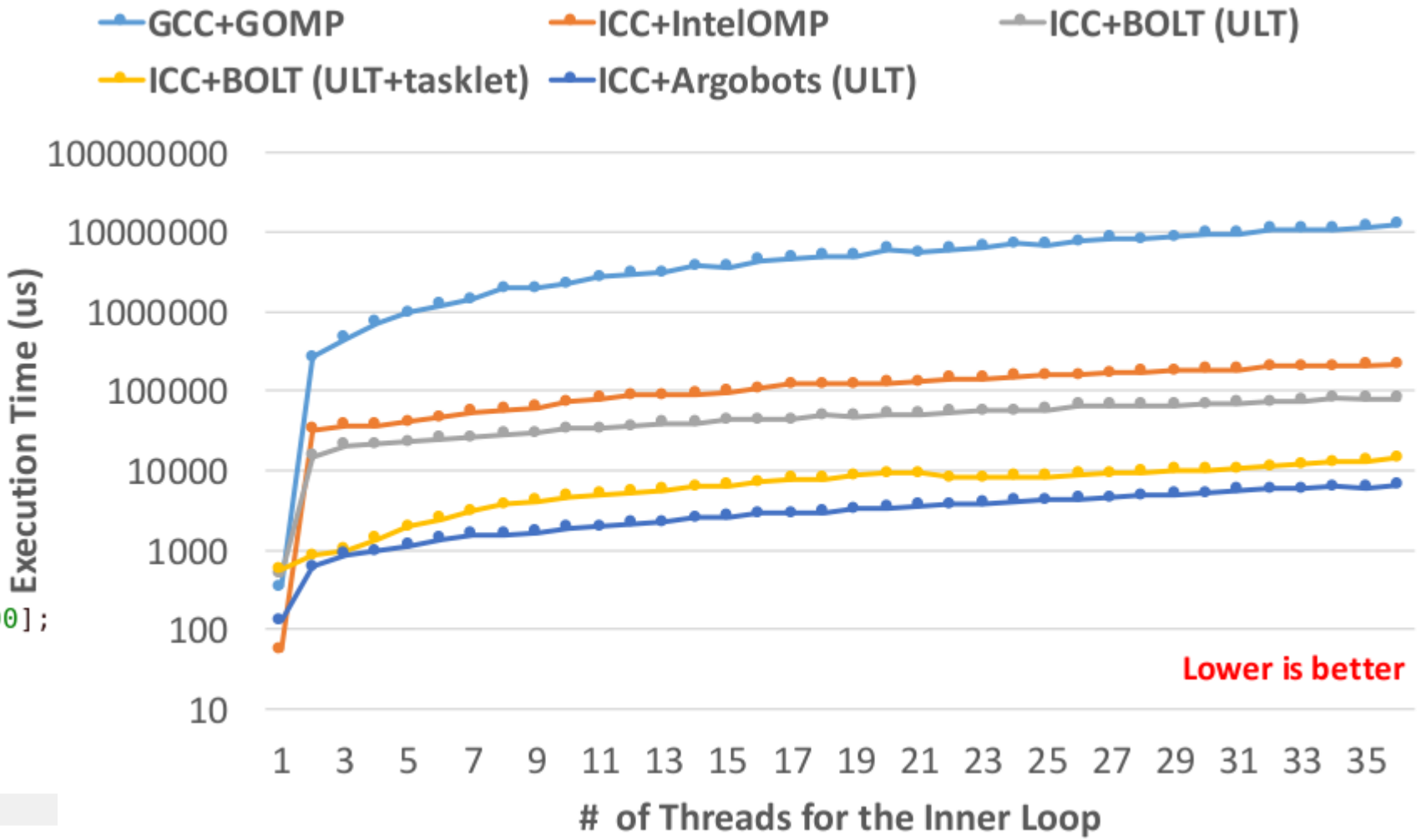

LLVM's runtime adapted to use our Argobots lightweight threading library.

http://www.bolt-omp.org/

# BOLT - “**B**OLT is **O**penMP over **L**ightweight **T**hreads” (Now Part of SOLLVE)

BOLT beats other runtimes by at least 10x on this nested parallelism benchmark.

Critical use case for composibility!



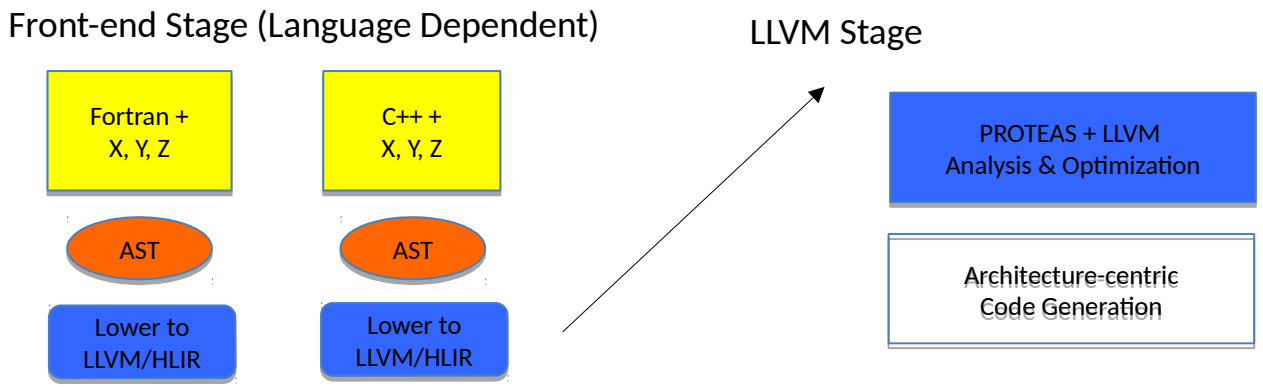

```
int in[1000][1000], out[1000][1000];

#pragma omp parallel for
  for (i = 0; i < 1000; i++)
    lib_compute(i);

void lib_compute(int x) {
#pragma omp parallel for
  for (j = 0; j < 1000; j++)
    out[x][j] = compute(in[x][j]);
}
```



http://www.openmp.org/wp-content/uploads/2016-11-15-Sangmin_Seo-SC16_OpenMP.pdf

# PROTEAS – "**PRO**gramming **T**oolchain for **E**merging **A**rchitectures and **S**ystems"

- Developing IR-level representations of parallelism constructs.
- Implementing optimizations on those representations to enable performance-portable programming.
- Exploring how to expose other aspects of modern memory hierarchies (such as NVM).



# (Compiler) Optimizations for OpenMP Code

OpenMP is already an abstraction layer. Why can't programmers just write the code optimally?

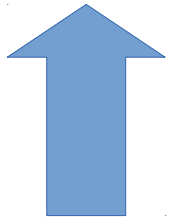
- Because what is optimal is different on different architectures.
- Because programmers use abstraction layers and may not be able to write the optimal code directly:

```
in library1:
void foo() {
  std::for_each(std::execution::par_unseq, vec1.begin(), vec1.end(), ...);
}

in library2:
void bar() {
  std::for_each(std::execution::par_unseq, vec2.begin(), vec2.end(), ...);
}

foo(); bar();
```

# (Compiler) Optimizations for OpenMP Code

```
void foo(double * restrict a, double * restrict b, etc.) {
#pragma omp parallel for
      for (i = 0; i < n; ++i) {
            a[i]  = e[i]*(b[i]*c[i] + d[i]) + f[i];
            m[i] = q[i]*(n[i]*o[i] + p[i]) + r[i];
      }
}
```

Split the loop

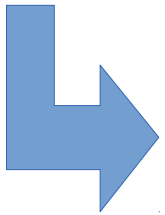


Or should we fuse instead?

```
void foo(double * restrict a, double * restrict b, etc.) {
#pragma omp parallel for
      for (i = 0; i < n; ++i) {
            a[i]  = e[i]*(b[i]*c[i] + d[i]) + f[i];
      }
#pragma omp parallel for
      for (i = 0; i < n; ++i) {
            m[i] = q[i]*(n[i]*o[i] + p[i]) + r[i];
      }
}
```

# (Compiler) Optimizations for OpenMP Code

```
void foo(double * restrict a, double * restrict b, etc.) {
#pragma omp parallel for
      for (i = 0; i < n; ++i) {
            a[i]  = e[i]*(b[i]*c[i] + d[i]) + f[i];
      }
#pragma omp parallel for
      for (i = 0; i < n; ++i) {
            m[i] = q[i]*(n[i]*o[i] + p[i]) + r[i];
      }
}
```



(we might want to fuse
the parallel regions)

```
void foo(double * restrict a, double * restrict b, etc.) {
#pragma omp parallel
  {
#pragma omp for
      for (i = 0; i < n; ++i) {
            a[i]  = e[i]*(b[i]*c[i] + d[i]) + f[i];
      }
#pragma omp for
      for (i = 0; i < n; ++i) {
            m[i] = q[i]*(n[i]*o[i] + p[i]) + r[i];
      }
  }
}
```

# (Compiler) Optimizations for OpenMP Code

In order to implement non-trivial parallelism optimizations, we need to move from "early outlining" to "late outlining."

The optimizer misses:

- Point aliasing information from the parent function
- Loop bounds (and other loop information) from the parent function
- And more…

But perhaps most importantly, it forces us to decide early **how** to lower the parallelism constructs. With some analysis first, after inlining, we can do a much better job (especially when targeting accelerators).

…ivalent of:

…_for_body(…) {

…allel_loop(&parallel_for_body, …);

Op… …does not know about the loo… …r the relationship between the code in the outlined body and the parent function.

# (Compiler) Optimizations for OpenMP Code

An example of where we might generate very different code after analysis...

```
#pragma omp target
{
  // This is a “serial” region on the device.
  foo();
  // So it this.
}

void foo() {
#pragma omp parallel for
  for (int I = 0; I < N; ++I) { … }
}
```

On a GPU, you launch some number of SIMT threads: there is no “serial” device execution. To support this general model, we need to generate a complex state machine in each GPU thread. This:

- Wastes resources
- Adds extra synchronization
- Increases register pressure

With late lowering, we could do an analysis to determine that there is no serial code in the parallel region and:

- Generate the (efficient) code the user expects.
- Analyze memory accesses and potentially use local/shared/texture memory.

# (Compiler) Optimizations for OpenMP Code

In order to implement non-trivial parallelism optimizations, we need to move from "early outlining" to "late outlining."

These markers are currently being designed. They might be intrinsics, perhaps also using operand bundles, but also require several special properties:

- alloca instructions inside the region must stay inside the region.
- The region markers must appear to capture/access pointers used inside the region (regions might run more than once, or after function returns, etc.).
- For loops, prevent the introduction of new loop-car… dependencies (duplicate induction v…
- UB if exception-triggered unwinding…

LVM IR equivalent of:

```
…{
@begin_parallel_for
for (…) {

…_for
```

…nt of:

```
…body(…) {

}

void foo() {
  __run_parallel_loop(&parallel_for_body, …);
}
```

If we don't also handle C++ lambdas using this kind of mechanism, we won't get the full benefit!

LLVM Optimizer

## ARES/HeteroIR – Predecessors to PROTEAS

- Developed a high-level IR targeted by OpenACC (and other models).
- http://ft.ornl.gov/research/openarc
- https://github.com/lanl/ares

Gives some idea of how much autotuning plus target-code-model guided transformations can help.

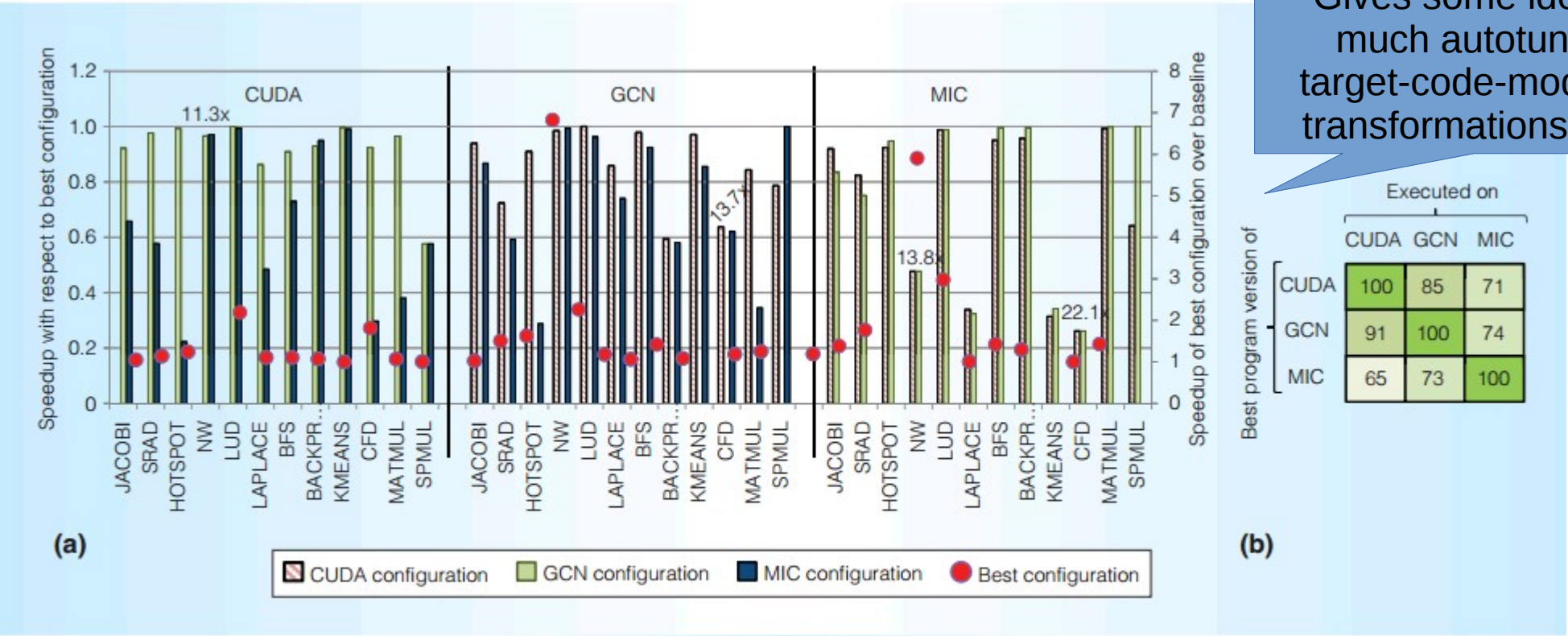
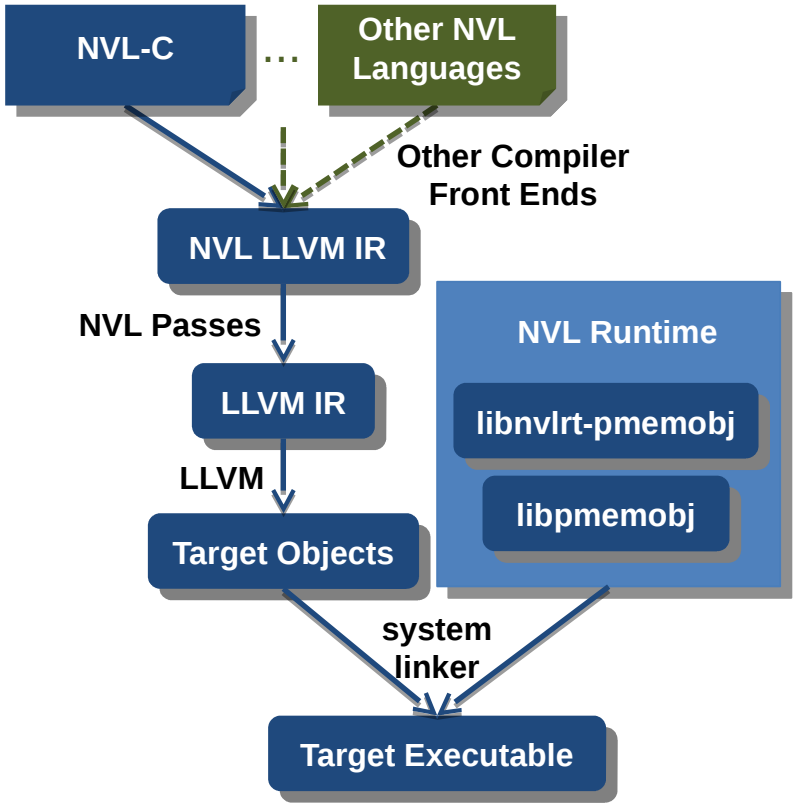



Figure 4. Performance portability of the OpenACC programming model as obtained by OpenARC. (a) Performance portability evaluation. (b) Performance portability achieved across benchmarks. Better performance portability is achieved among the GPU architectures.

http://ieeexplore.ieee.org/abstract/document/7155420/

# NVL-C – Predecessors to PROTEAS

- Experimenting with how to use NVM
- http://ft.ornl.gov/research/nvl-c




Extensions to C with transactions.

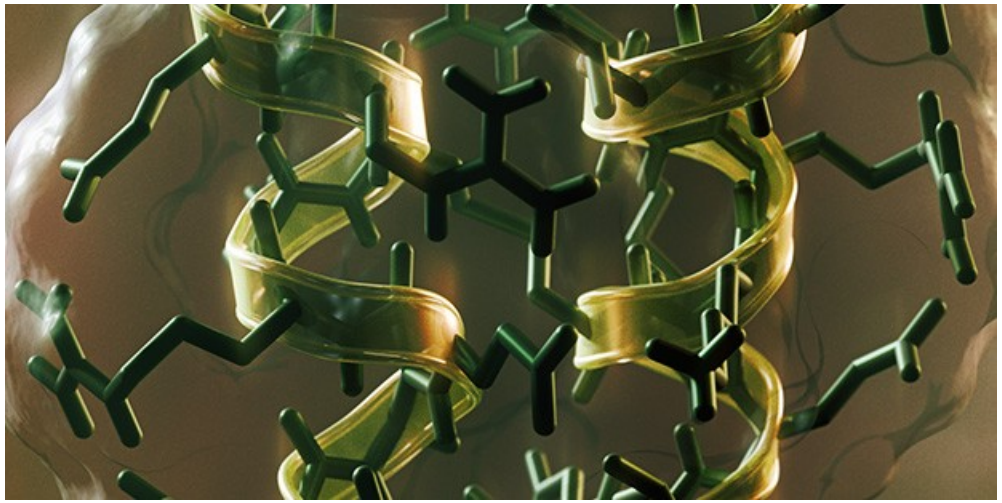
```
#include <nvl.h>
struct list {
  int value;
  nvl struct list *next;
};
void remove(int k) {
  nvl_heap_t *heap
    = nvl_open("foo.nvl");
  nvl struct list *a
    = nvl_get_root(heap, struct list);
  #pragma nvl atomic
  while (a->next != NULL) {
    if (a->next->value == k)
      a->next = a->next->next;
    else
      a = a->next;
  }
  nvl_close(heap);
}
```

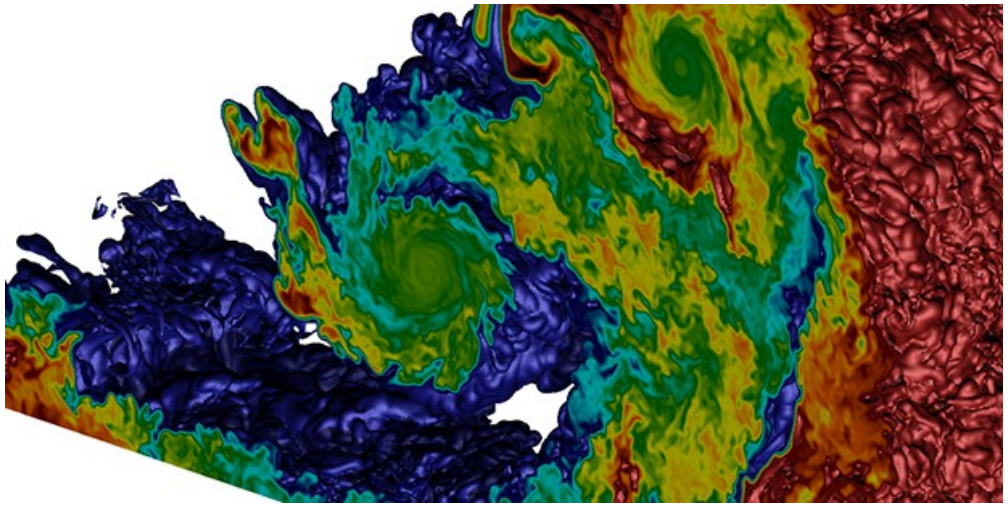
Will high-performance NVM fundamentally change the way that people write software?

(Work by Joel E. Denny, Seyong Lee, and Jeffrey S. Vetter

# In Conclusion...

- Future HPC hardware will be diverse.
- Work is needed on applications, abstraction libraries, and compilers (and related tools).
- Enhancing LLVM to understand parallelism provides an enabling underlying technology for performance portability!





# Acknowledgments


- The LLVM community (including our many contributing vendors)
- ALCF, ANL, and DOE
- ALCF is supported by DOE/SC under contract DE-AC02-06CH11357