



Lithe: Enabling Efficient Composition of Parallel Libraries

Heidi Pan, Benjamin Hindman, Krste Asanović

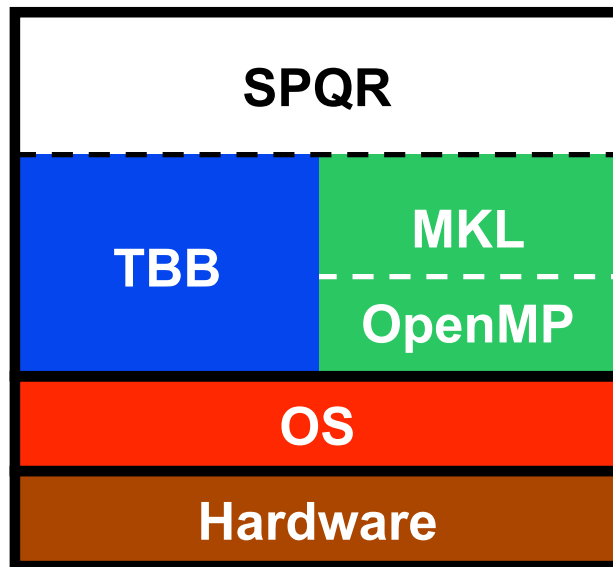
xoxo@mit.edu • {benh, krste}@eecs.berkeley.edu
Massachusetts Institute of Technology • UC Berkeley

ParLab Boot Camp • August 19, 2009

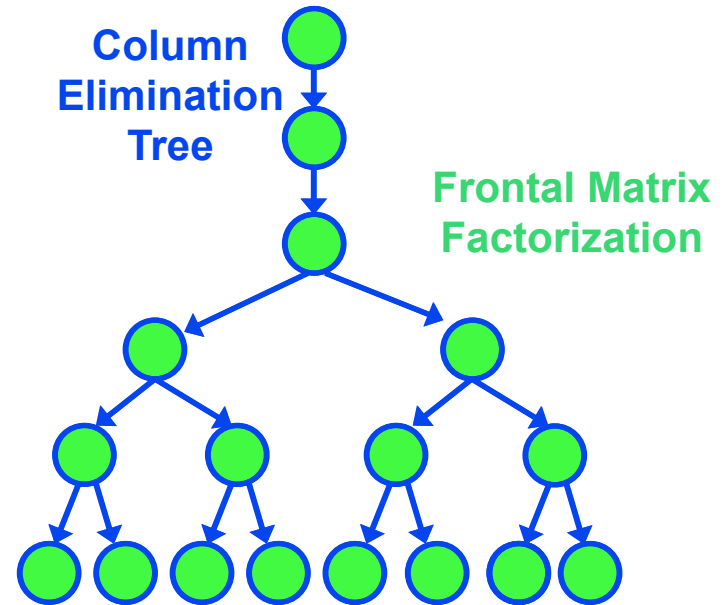
Real-World Parallel Composition Example

Sparse QR Factorization

(Tim Davis, Univ of Florida)



System Stack

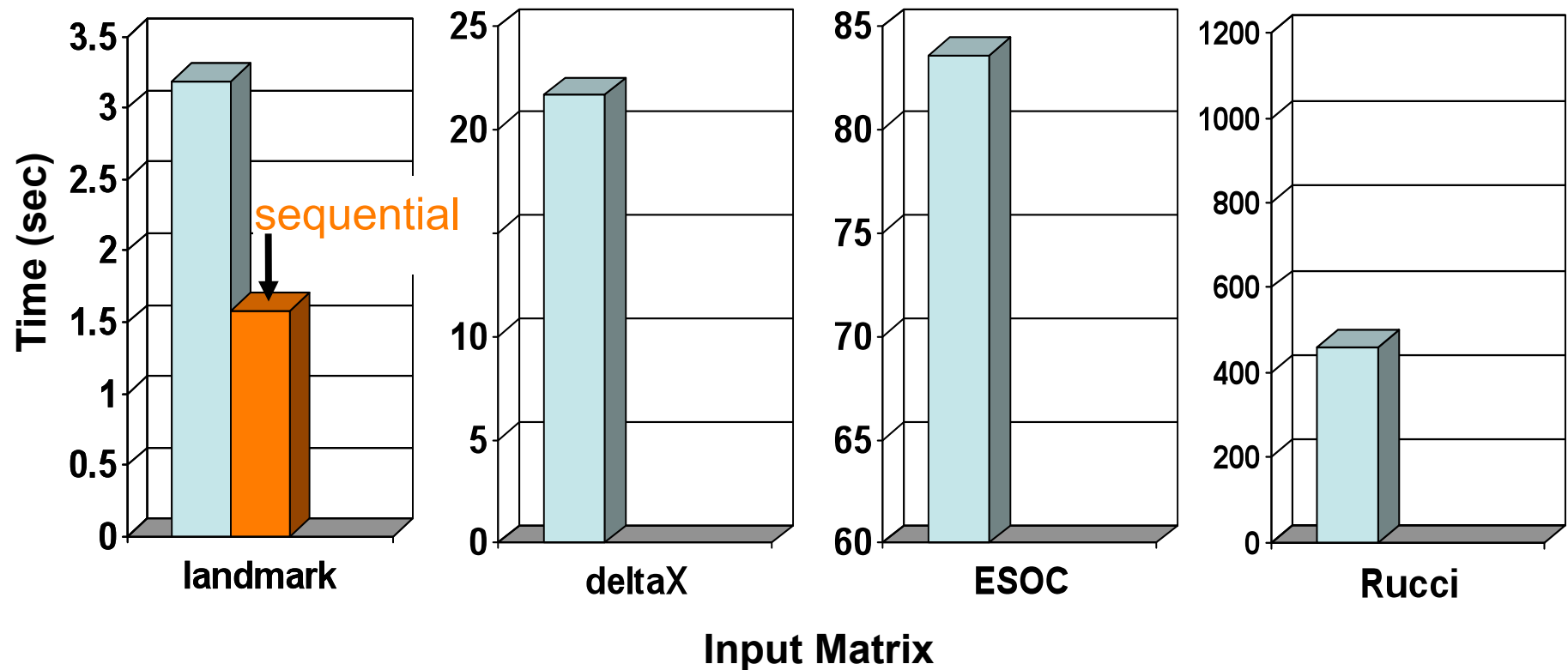


Software Architecture

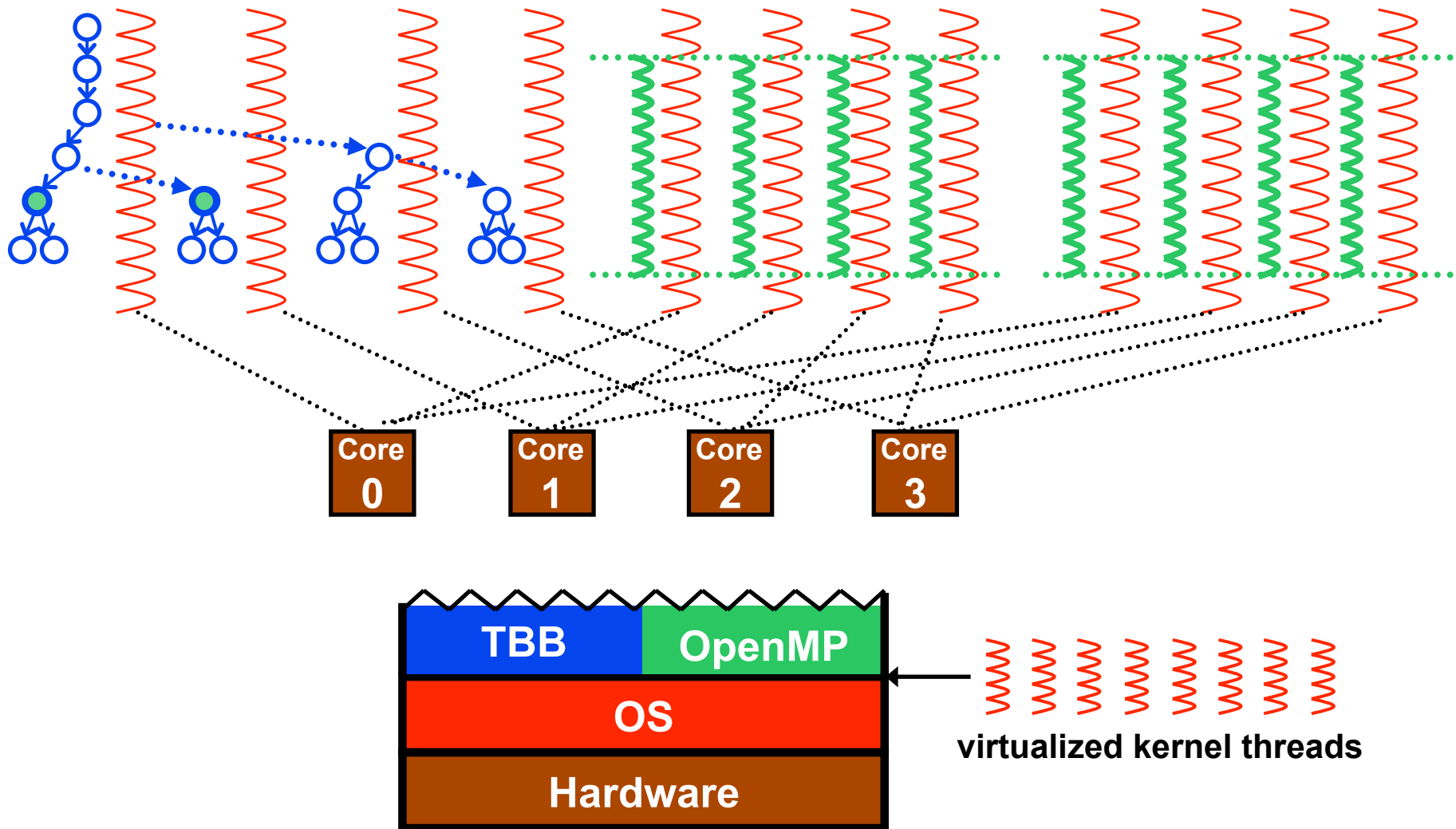
Out-of-the-Box Performance

Performance of SPQR on 16-core Machine

Out-of-the-Box



Out-of-the-Box Libraries Oversubscribe the Resources



MKL Quick Fix

Using Intel MKL with Threaded Applications

<http://www.intel.com/support/performance/tools/libraries/mkl/sb/CS-017177.htm>

Software Products

Intel® Math Kernel Library (Intel® MKL)
Using Intel® MKL with Threaded Applications

Page Contents:

- Memory Allocation MKL: Memory appears to be allocated and not released when calling some Intel MKL routines (e.g. sgtrf).
- Using Threading with BLAS and LAPACK
- Setting the Number of Threads
- Changing the Number of Threads
- Can I use Intel MKL if I thread my application?

Memory Allocation MKL
When you use some Intel® MKL routines, memory is allocated and not released. One of the advantages of using OpenMP® is that it requires even for single-processor systems to occur once the first time the allocation persists until the application will allocate a stack equal to of memory that is automatically allocations and the number of

Using Threading with BLAS
Intel MKL is threaded in a number of Level 3 BLAS, LAPACK, and FFT situations in which conflicts of We list them here with recommendations the problem exists is appropriate.

If the user threads the program using OpenMP directives and uses the Intel® Compilers to compile the program, Intel MKL and the user program will both use the same threading library. Intel MKL tries to determine if it is in a parallel region in the program, and if it is, it does not spread its operations over multiple threads. But Intel MKL can be aware that it is in a parallel region only if the threaded program and Intel MKL are using the same threading library. If the user program is threaded by some other means, Intel MKL may operate in multithreaded mode and the computations may be corrupted. Here are several cases and our recommendations:

- User threads the program using OS threads (pthreads on Linux®, Win32® threads on Windows®). If more than one thread calls Intel MKL and the function being called is threaded, it is important that threading in Intel MKL be turned off. Set `OMP_NUM_THREADS=1` in the environment.
- User threads the program using OpenMP directives and/or pragmas and compiles the program using a compiler other than a compiler from Intel. This is more problematic because setting `OMP_NUM_THREADS` in the environment affects both the compiler's threading library and the threading

library with Intel MKL. In this case, the safe approach is to set `OMP_NUM_THREADS=1`.

- Multiple programs are running on a multiple-CPU system. In cluster applications, the parallel program can run separate instances of the program on each processor. However, the threading software will see multiple processors on the system even though each processor has a separate process running on it. In this case `OMP_NUM_THREADS` should be set to 1.
- If the variable `OMP_NUM_THREADS` environment variable is not set, then the default number of threads will be assumed 1.

Setting the Number of Threads for OpenMP® (OMP)

```
void main(int argc, char *argv){  
    double *a, *b, *c;  
    a = new double [SIZE*SIZE];  
    b = new double [SIZE*SIZE];  
    c = new double [SIZE*SIZE];  
  
    double alpha=1, beta=1;  
    int m=SIZE, n=SIZE, l=SIZE, lda=SIZE, ldb=SIZE, ldo=SIZE, i=0, j=0;  
    char transa='n', transb='n';  
  
    for (i=0; i<SIZE; i++){  
        for (j=0; j<SIZE; j++){  
            a[i*SIZE+j]= (double)(i+j);  
            b[i*SIZE+j]= (double)(i*j);  
            c[i*SIZE+j]= (double)0;  
        }  
    }  
    cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,  
               m, n, k, alpha, a, lda, b, ldb, beta, c, ldo);  
}
```

```
printf("row\ta\to\n");  
for ( i=0;i<10;i++){  
    printf("%d:\t%f\t%f\n", i, a[i*SIZE], c[i*SIZE]);  
}  
  
omp_set_num_threads(1);  
  
for (i=0; i<SIZE; i++){  
    for (j=0; j<SIZE; j++){  
        a[i*SIZE+j]= (double)(i+j);  
        b[i*SIZE+j]= (double)(i*j);  
        c[i*SIZE+j]= (double)0;  
    }  
}
```

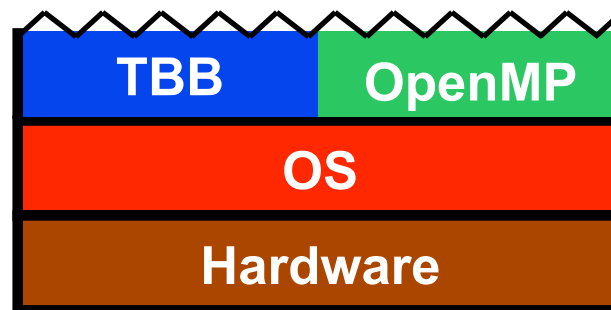
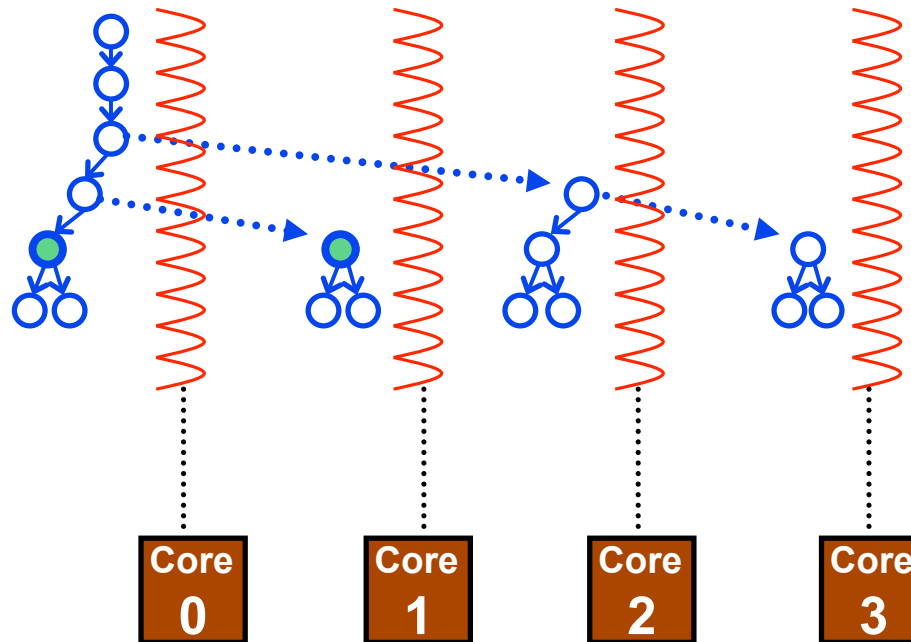
```
printf("row\ta\to\n");  
for ( i=0;i<10;i++){  
    printf("%d:\t%f\t%f\n", i, a[i*SIZE],  
c[i*SIZE]);  
}  
  
delete [] a;  
delete [] b;  
delete [] c;  
}
```

Can I use Intel MKL if I thread my application?

The Intel Math Kernel Library is designed and compiled for thread safety so it can be called from programs that are threaded. Calling Intel MKL routines that are threaded from multiple application threads can lead to conflict (including incorrect answers or program failures), if the calling library differs from the Intel MKL threading library.

• If more than one thread calls Intel MKL and the function being called is threaded, it is important that threading in Intel MKL be turned off. Set **OMP_NUM_THREADS=1** in the environment.

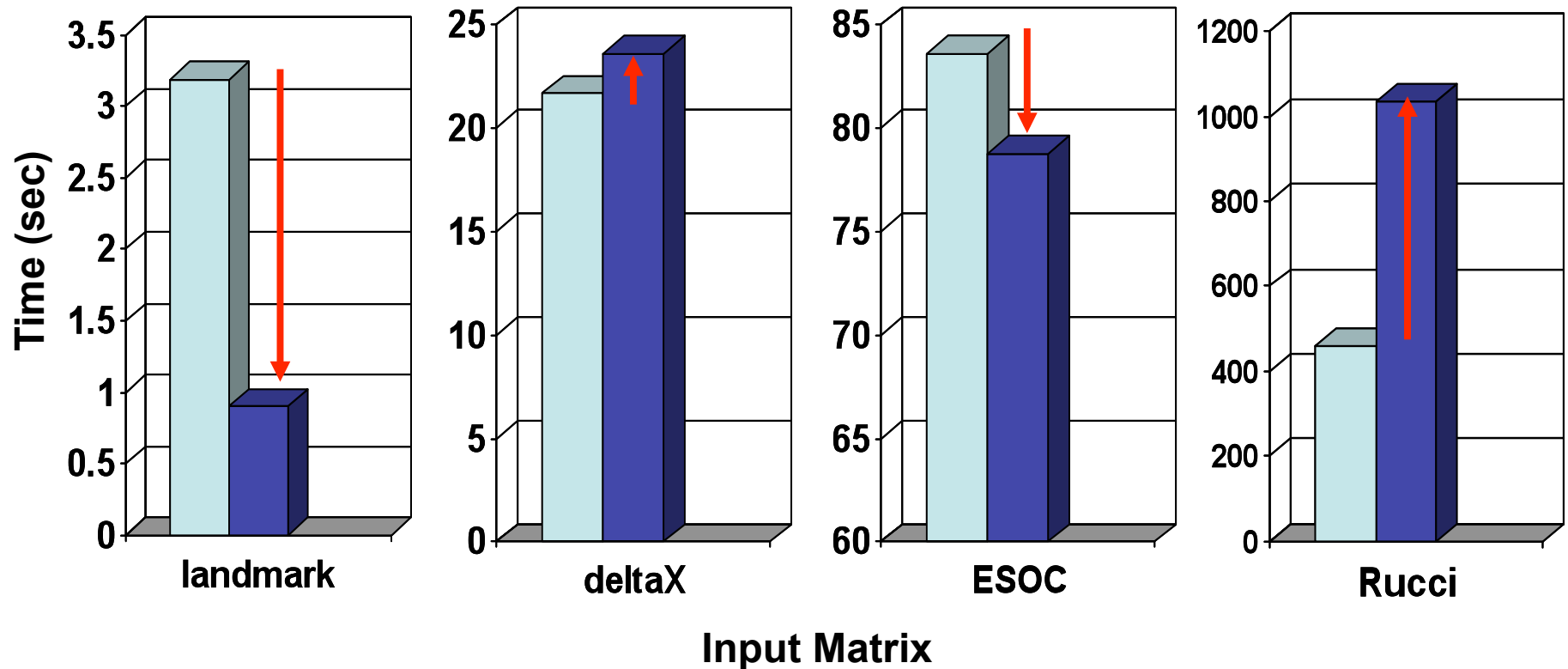
Sequential MKL in SPQR



Sequential MKL Performance

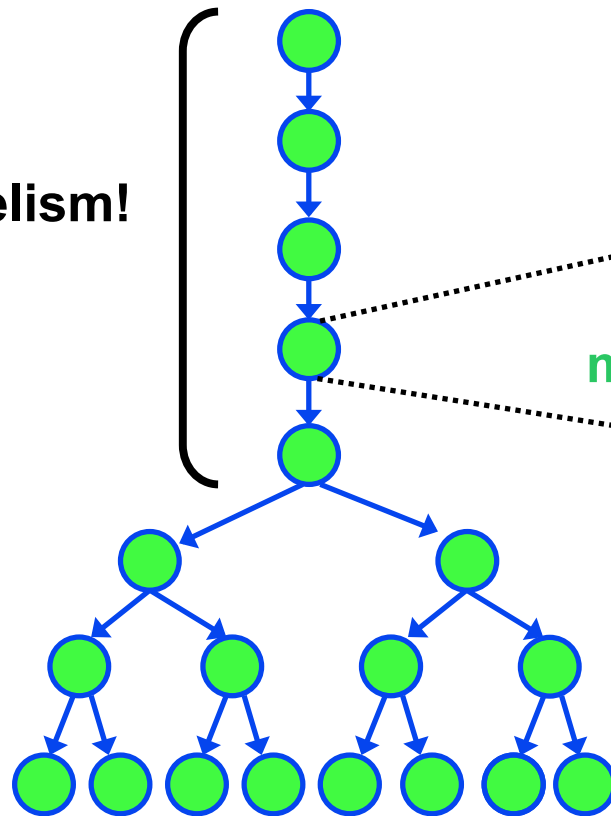
Performance of SPQR on 16-core Machine

Out-of-the-Box Sequential MKL



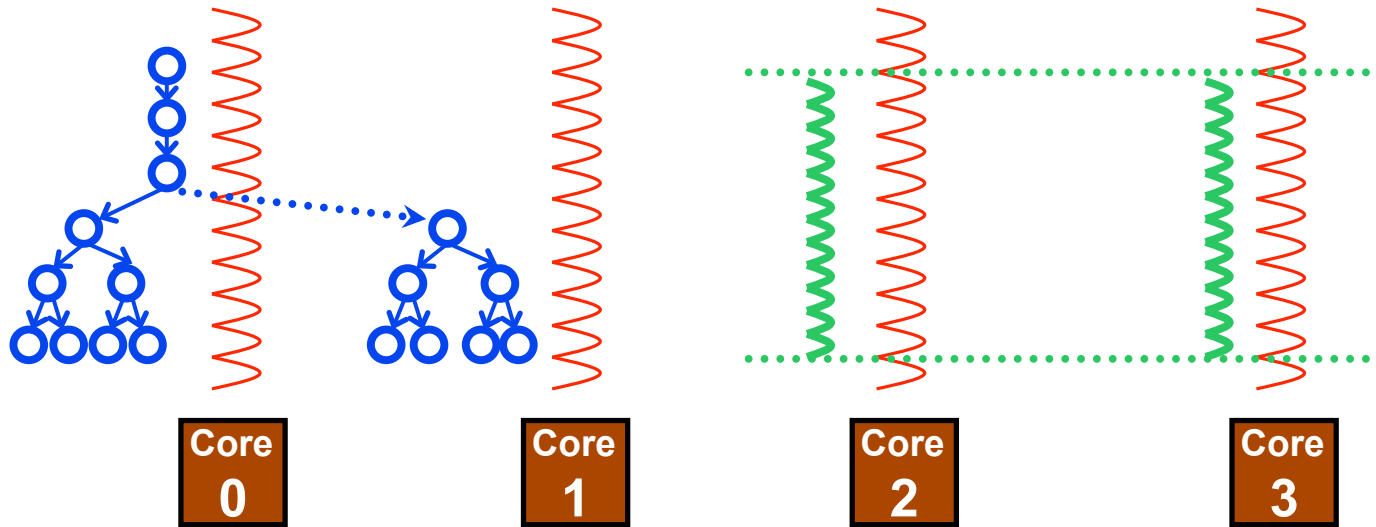
SPQR Wants to Use Parallel MKL

No **task-level** parallelism!



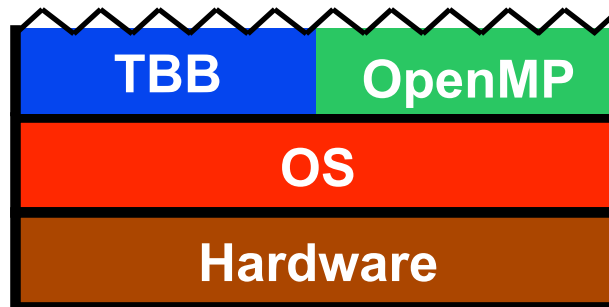
Want to exploit **matrix-level** parallelism.

Share Resources Cooperatively



`TBB_NUM_THREADS = 2`

`OMP_NUM_THREADS = 2`

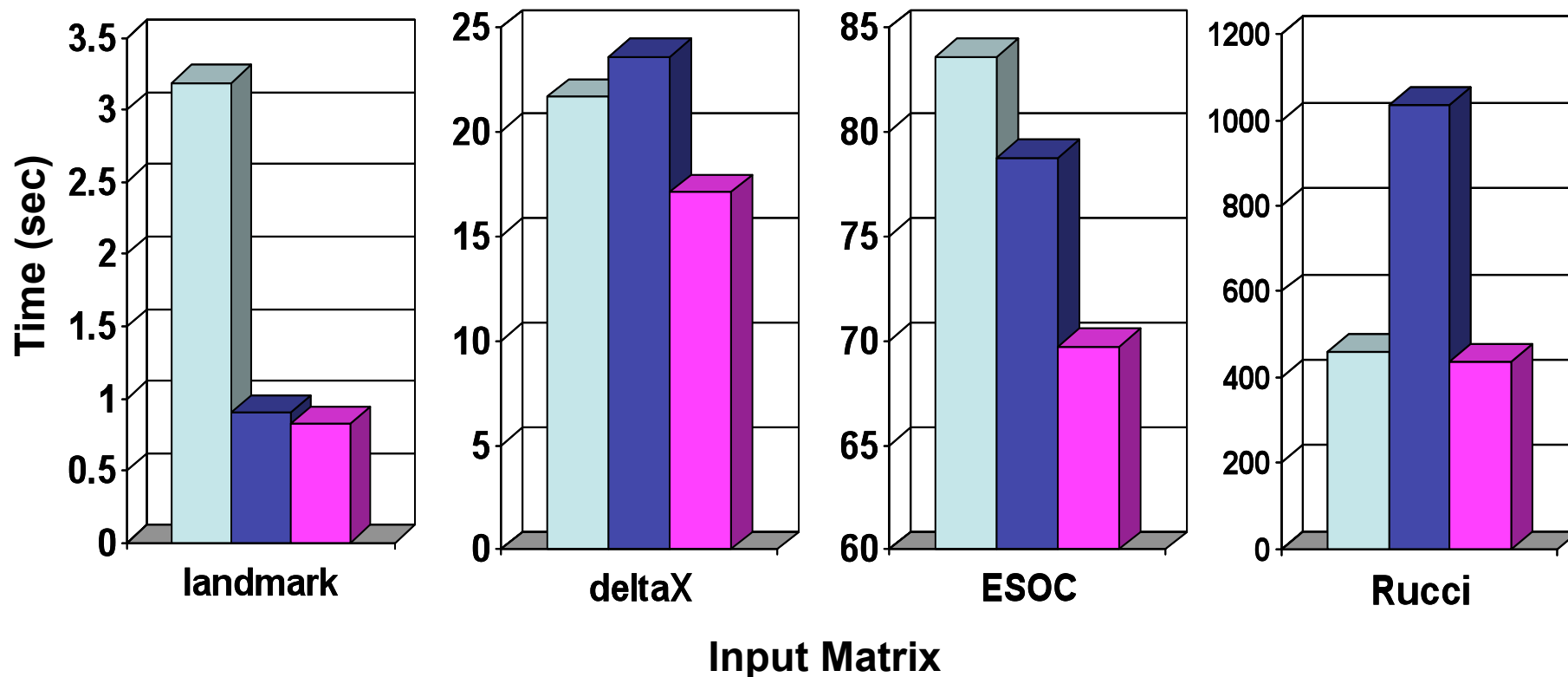


Tim Davis manually tunes libraries to effectively partition the resources.

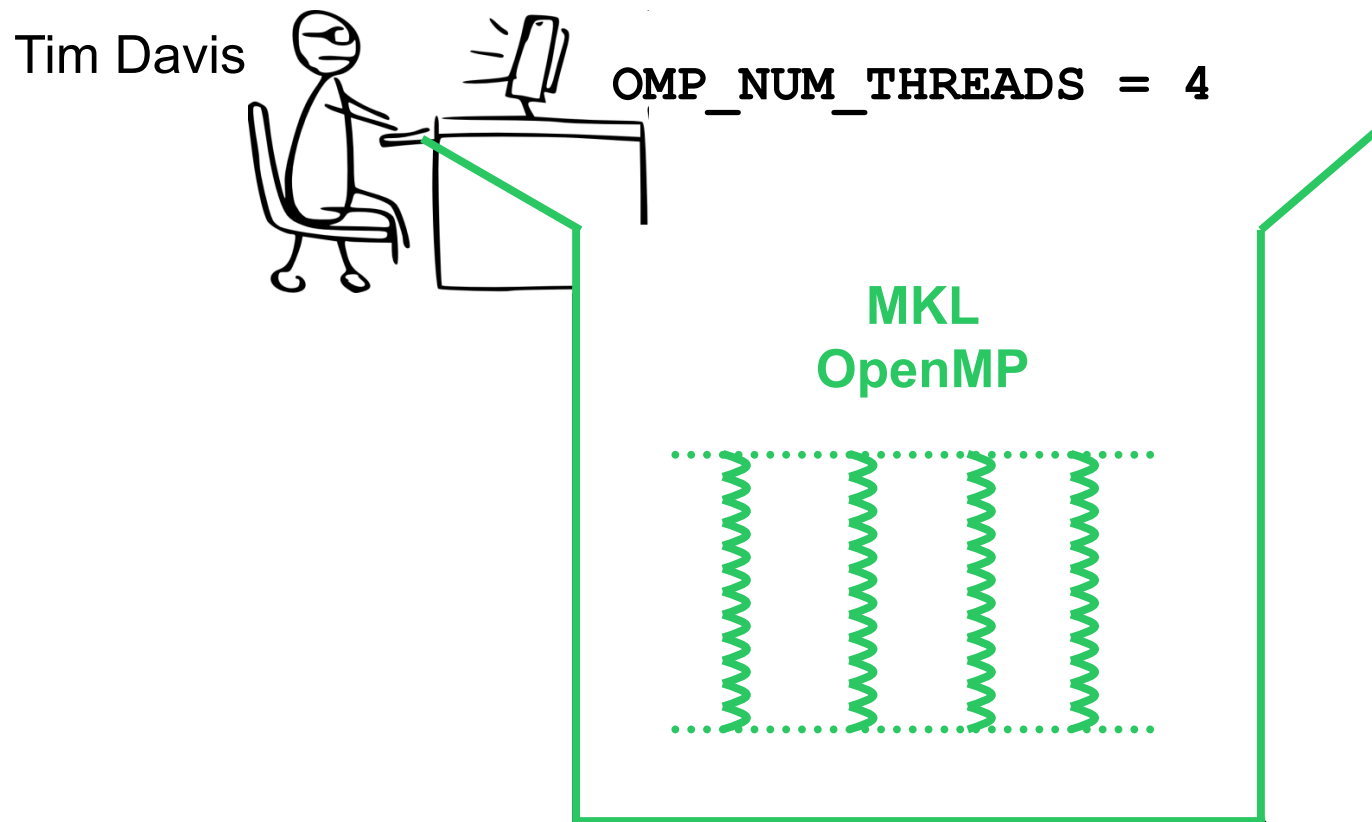
Manually Tuned Performance

Performance of SPQR on 16-core Machine

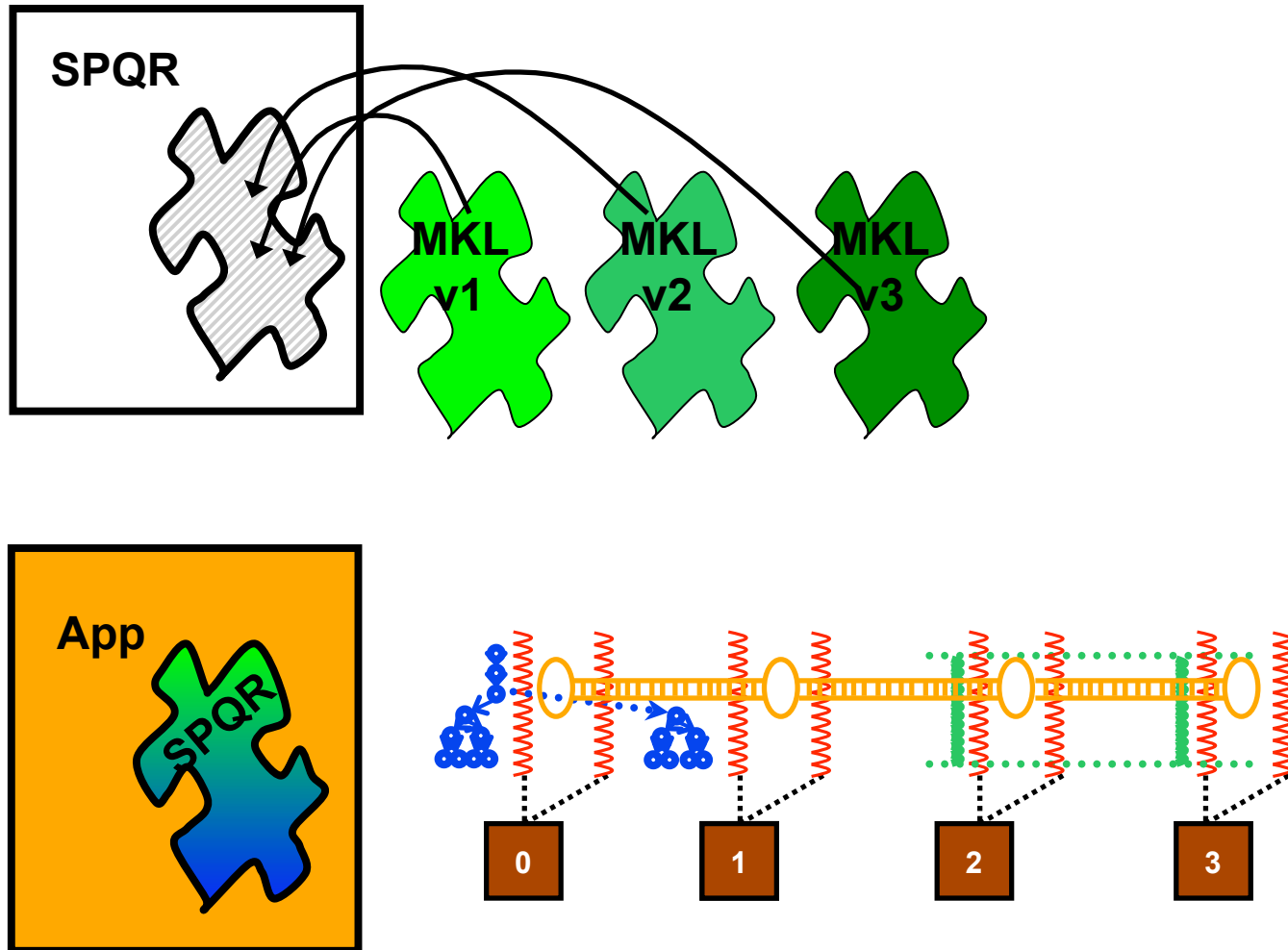
Out-of-the-Box Sequential MKL Manually Tuned



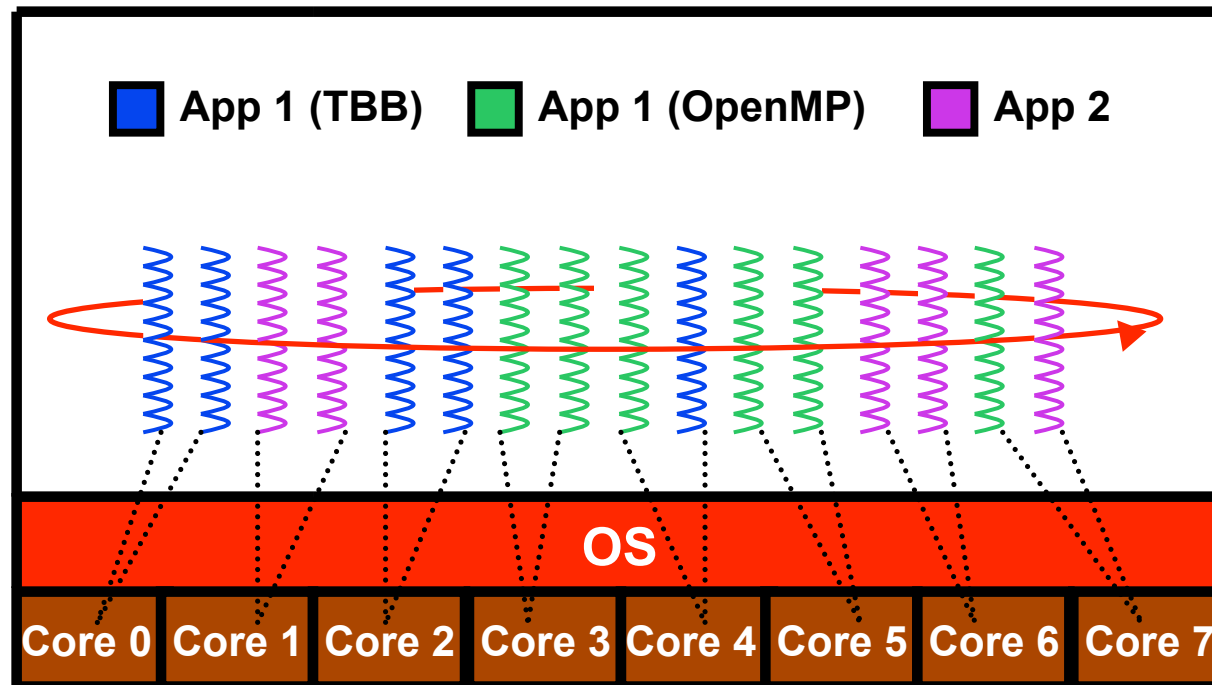
Manual Tuning Destroys Black Box Abstractions



Manual Tuning Destroys Code Reuse and Modular Updates

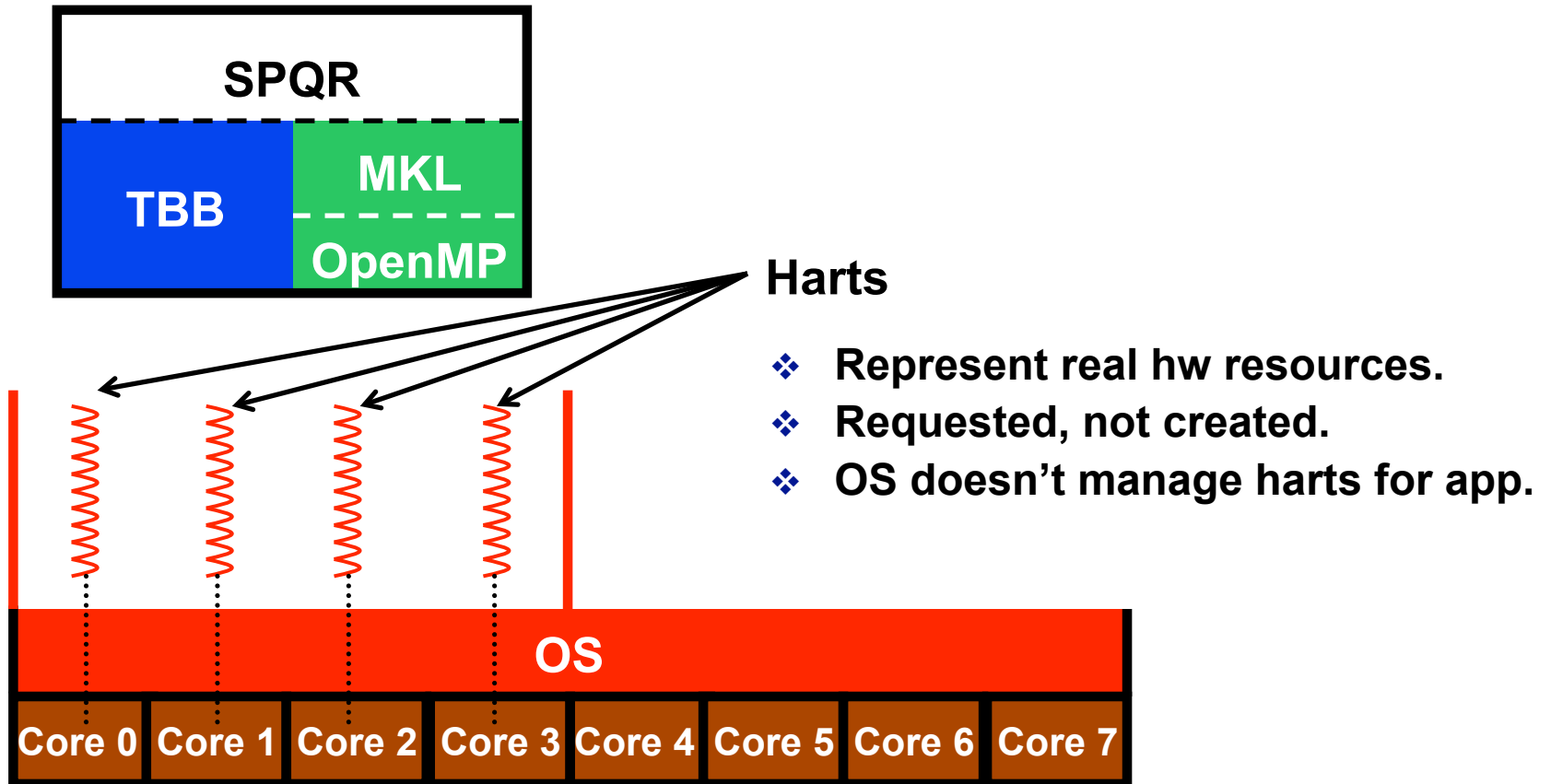


Virtualized Threads are Bad

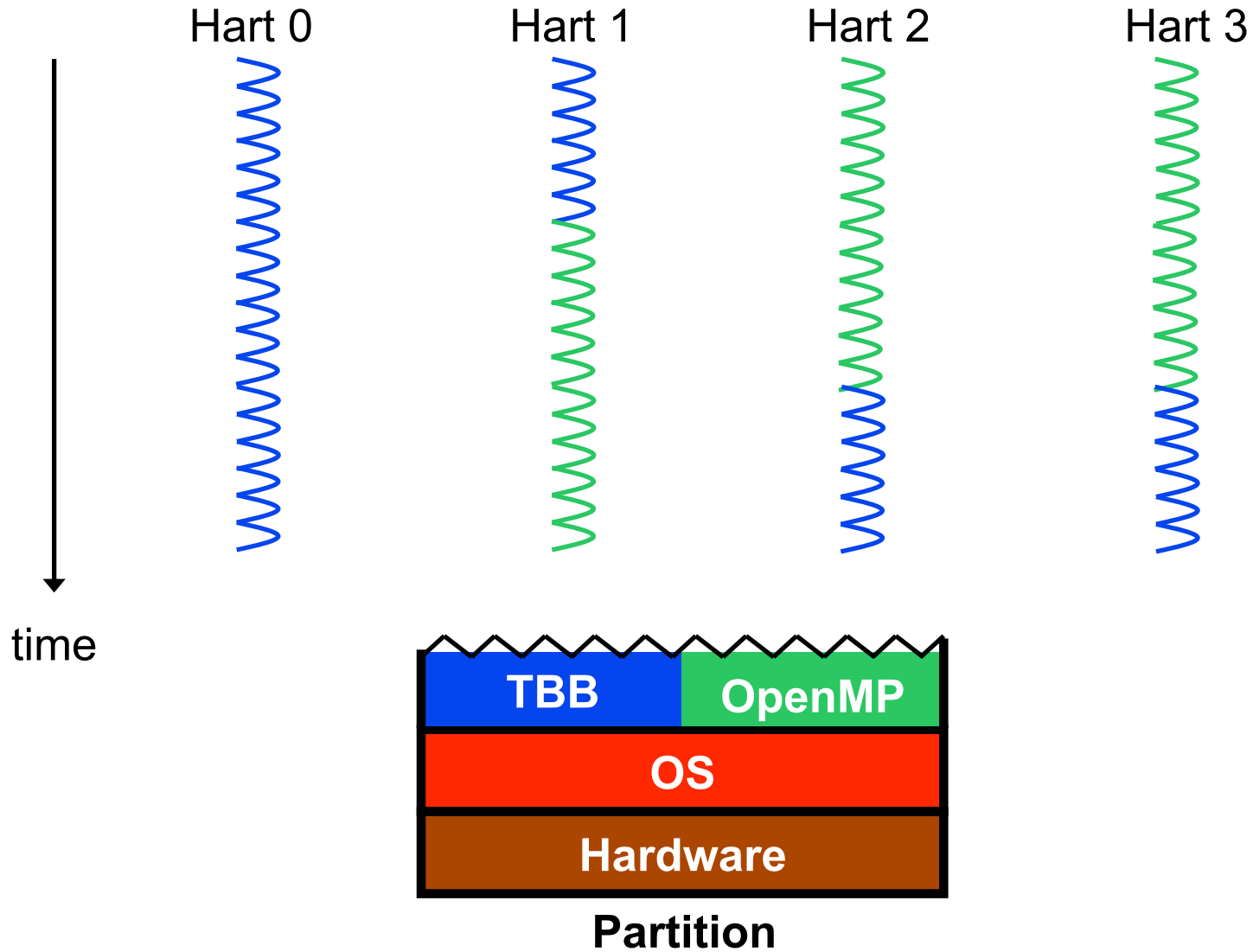


Different codes compete unproductively for resources.

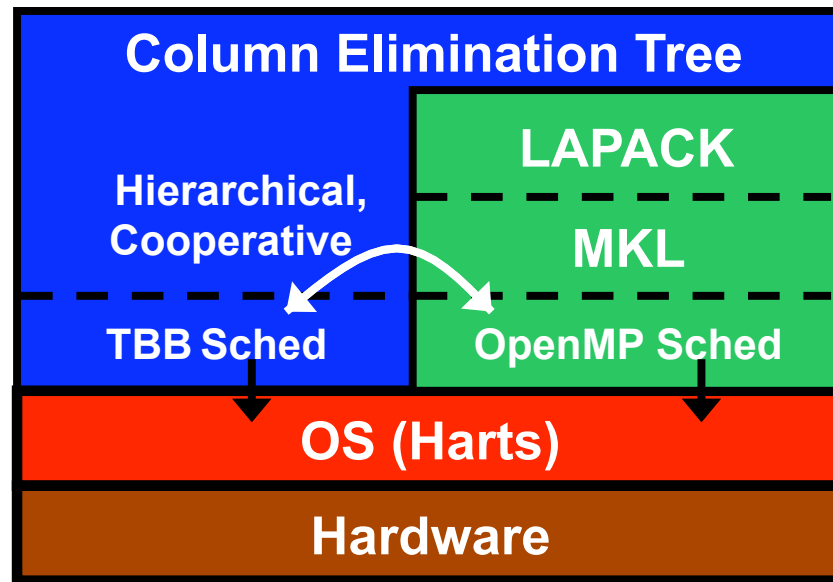
Harts: Hardware Thread Contexts



Sharing Harts

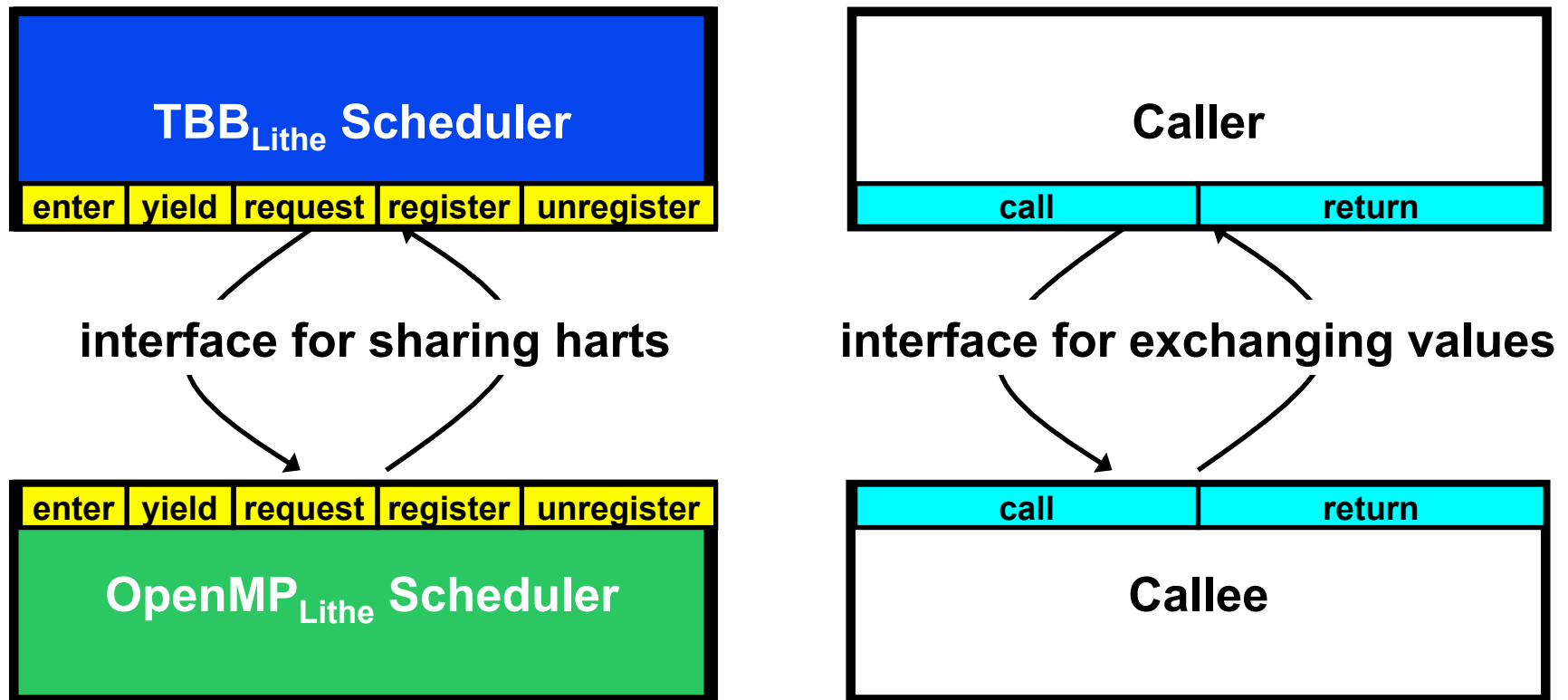


Hierarchical Cooperative Scheduling



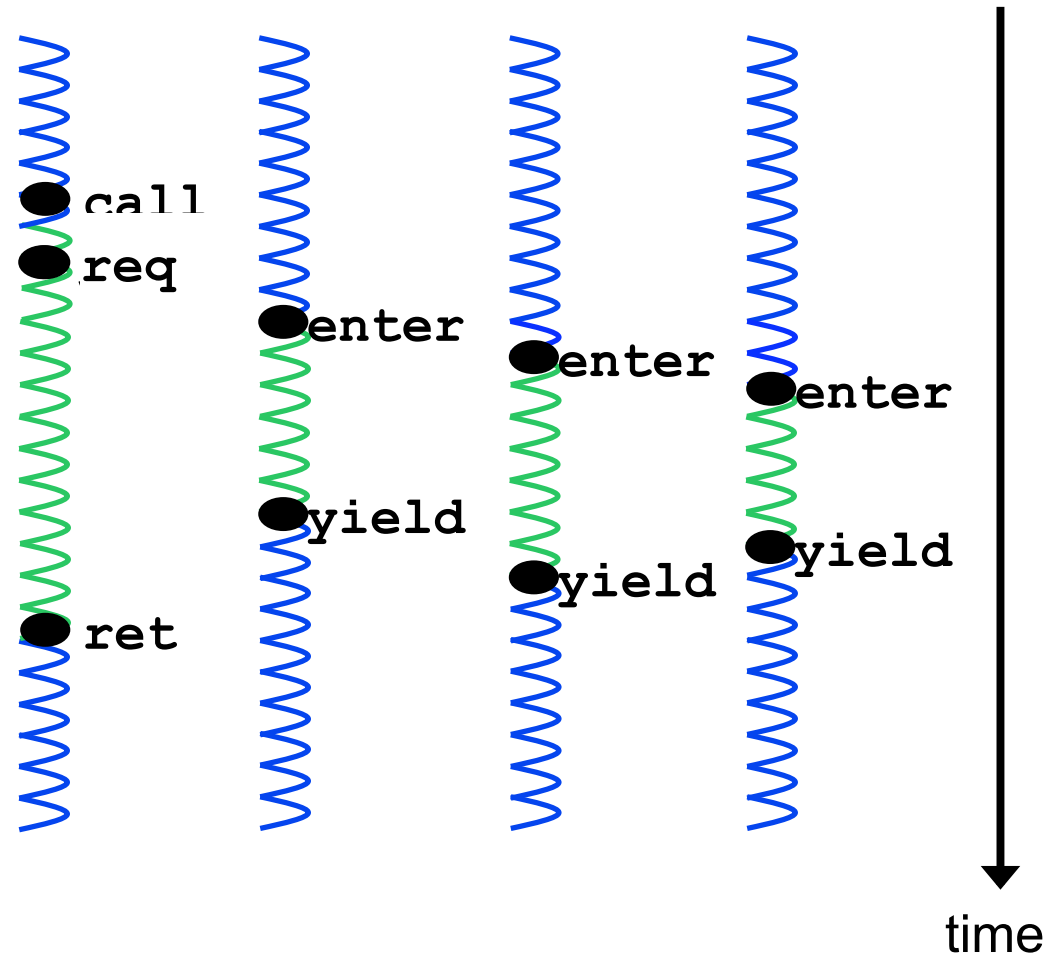
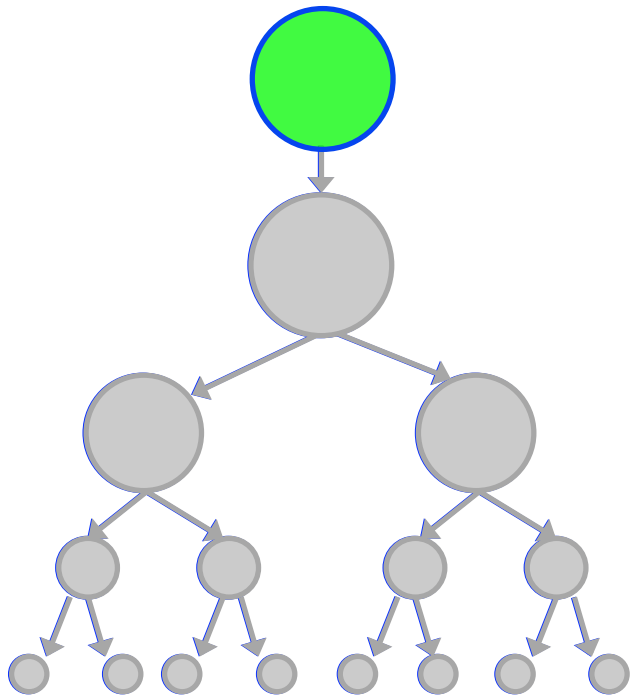
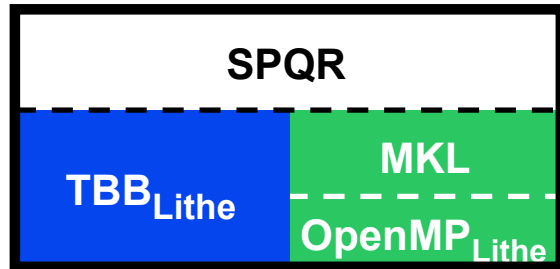
**Direct Control
of Resources**

Standard Lithe ABI

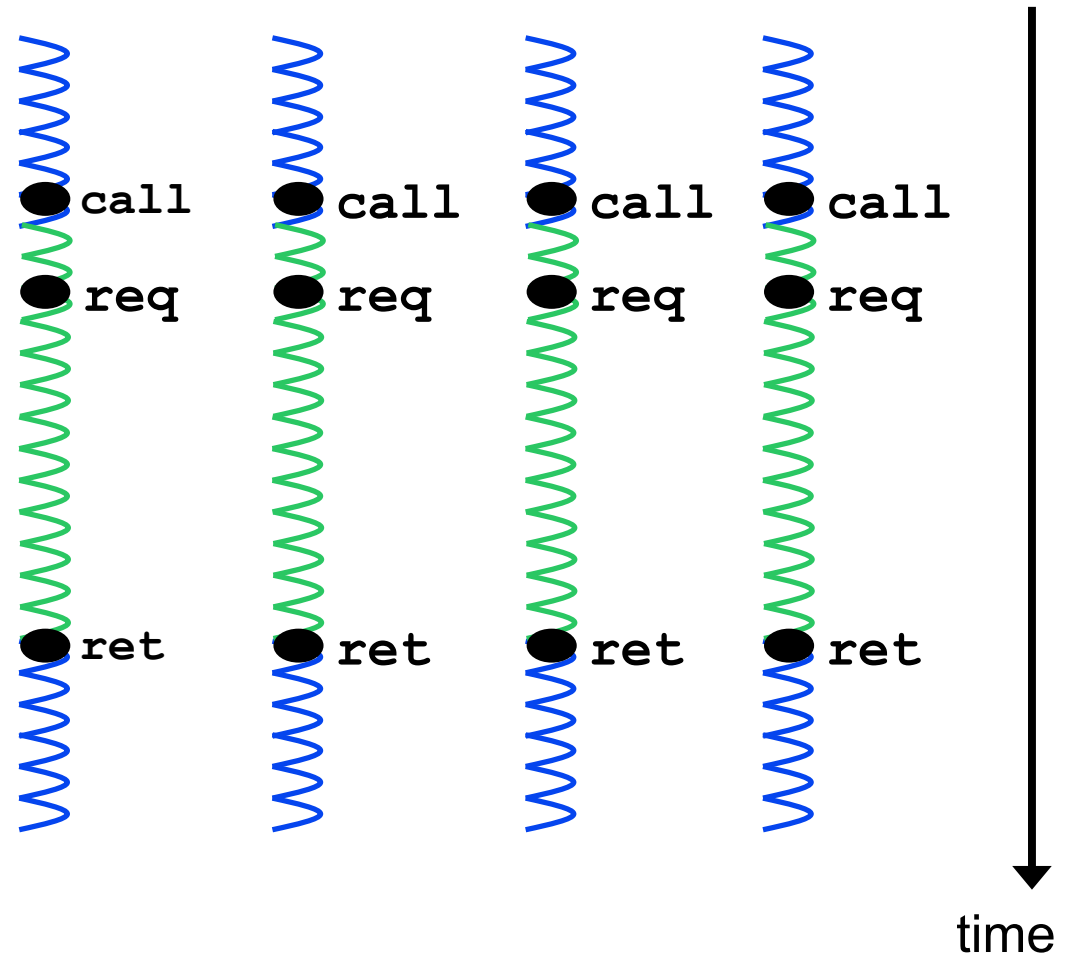
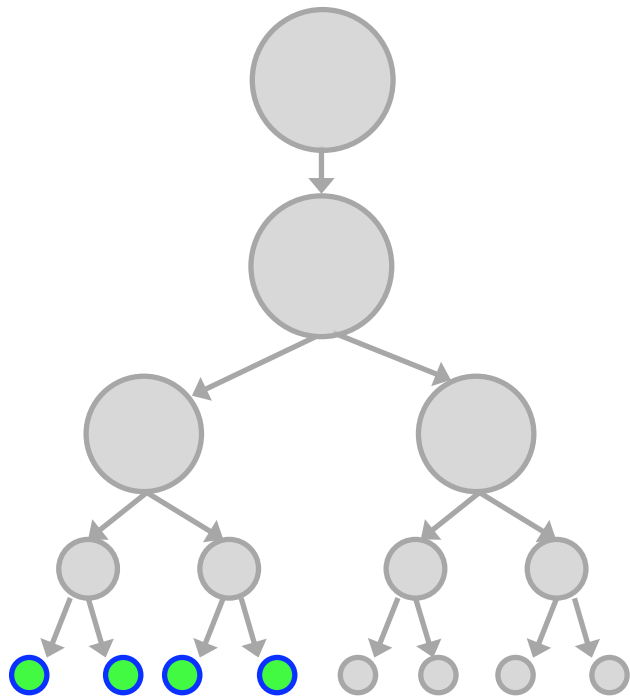
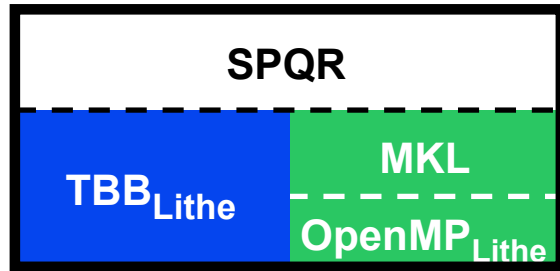


- ❖ Analogous to function call ABI for enabling interoperable codes.
- ❖ Mechanism for sharing harts, *not* policy.

SPQR with Lithe

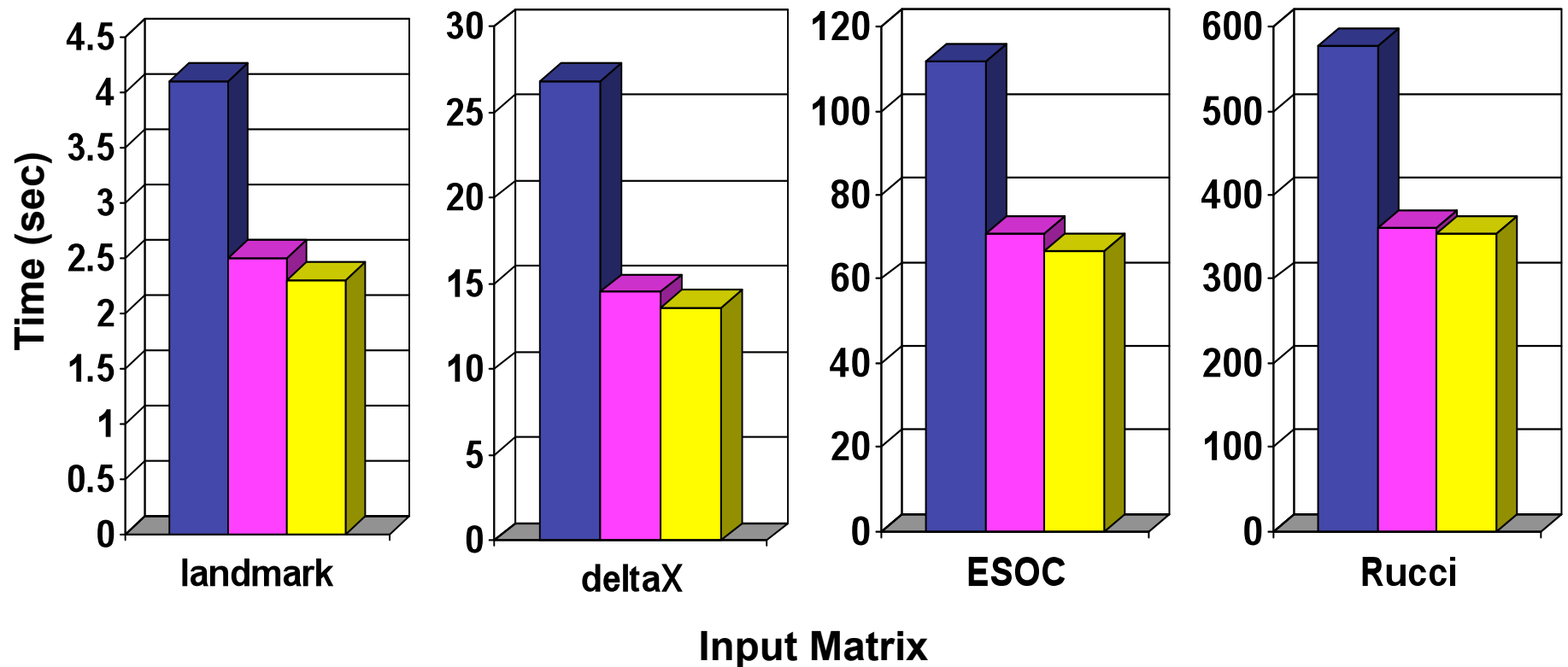


SPQR with Lithe

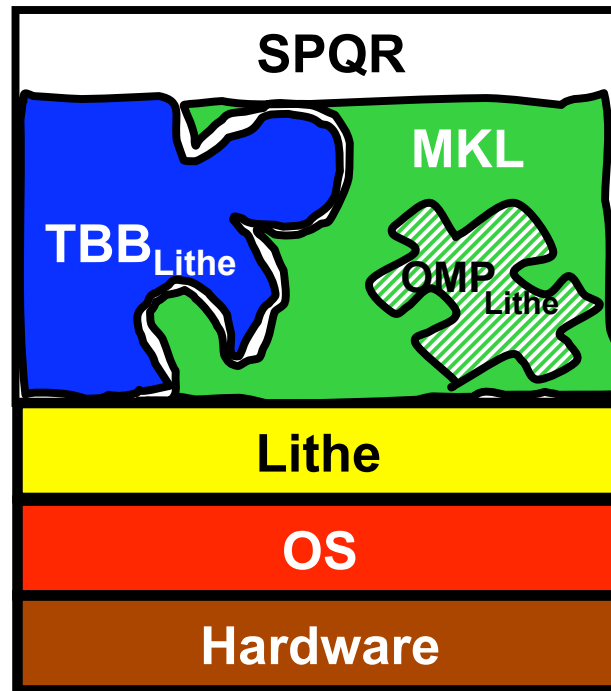


Performance of SPQR with Lithe

■ Out-of-the-Box ■ Manually Tuned ■ Lithe



Questions?



Lithe: Enabling Efficient Composition of Parallel Libraries

Acknowledgements

We would like to thank George Necula and the rest of Berkeley Par Lab for their feedback on this work.

Research supported by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227). This work has also been in part supported by a National Science Foundation Graduate Research Fellowship. Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation. The authors also acknowledge the support of the Gigascale Systems Research Focus Center, one of five research centers funded under the Focus Center Research Program, a Semiconductor Research Corporation program.

Microsoft[®]

