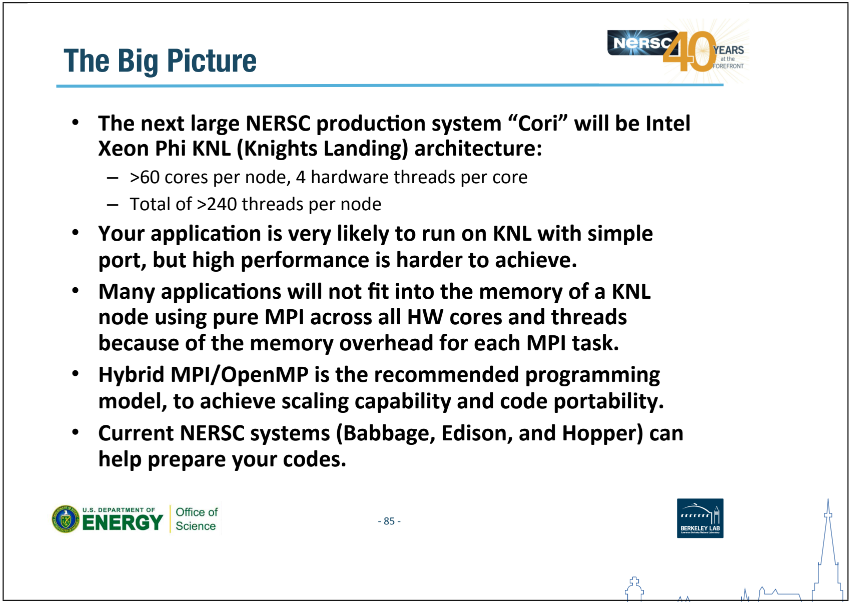
To thread or not to thread?
Why PETSc favors MPI-only

Plenary Discussion

PETSc User Meeting 2016

Based on:
MS35 - To Thread or Not To Thread
April 13, 2016
SIAM PP, Paris

The Big Picture



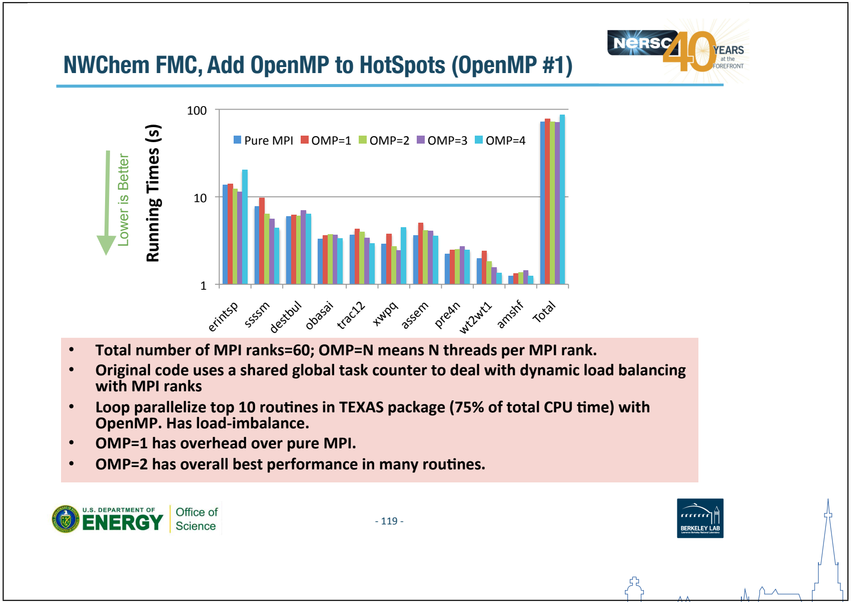
The Big Picture

- The next large NERSC production system "Cori" will be Intel Xeon Phi KNL (Knights Landing) architecture:
  - >60 cores per node, 4 hardware threads per core
  - Total of >240 threads per node
- Your application is very likely to run on KNL with simple port, but high performance is harder to achieve.
- Many applications will not fit into the memory of a KNL node using pure MPI across all HW cores and threads because of the memory overhead for each MPI task.
- Hybrid MPI/OpenMP is the recommended programming model, to achieve scaling capability and code portability.
- Current NERSC systems (Babbage, Edison, and Hopper) can help prepare your codes.

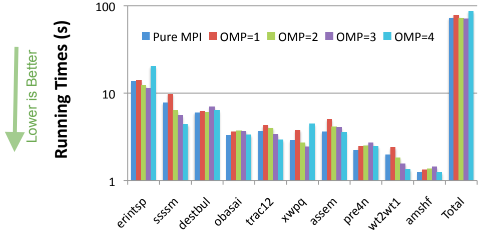
- 85 -

"OpenMP Basics and MPI/OpenMP Scaling", Yun He, NERSC, 2015

The Big Picture



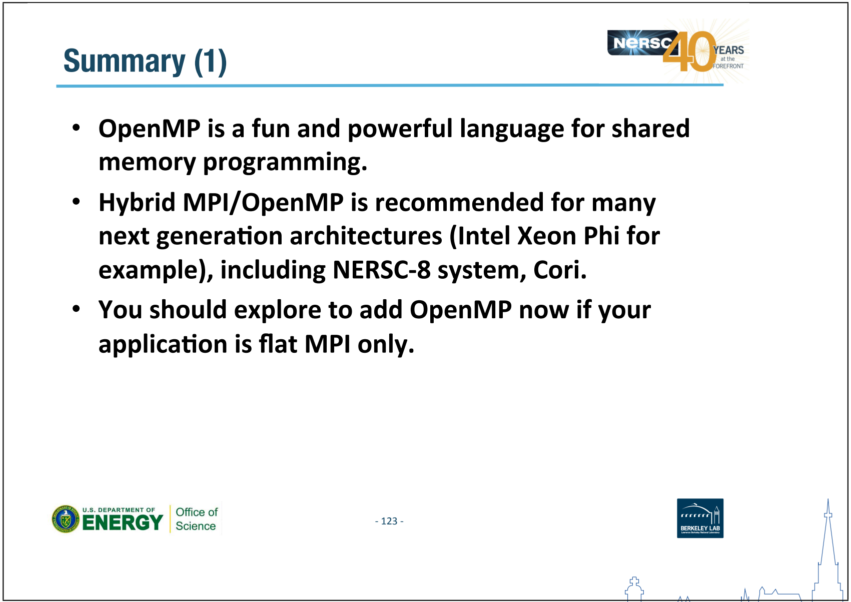
NWChem FMC, Add OpenMP to HotSpots (OpenMP #1)



- Total number of MPI ranks=60; OMP=N means N threads per MPI rank.
- Original code uses a shared global task counter to deal with dynamic load balancing with MPI ranks
- Loop parallelize top 10 routines in TEXAS package (75% of total CPU time) with OpenMP. Has load-imbalance.
- OMP=1 has overhead over pure MPI.
- OMP=2 has overall best performance in many routines.

“OpenMP Basics and MPI/OpenMP Scaling”, Yun He, NERSC, 2015

The Big Picture



Summary (1)

- OpenMP is a fun and powerful language for shared memory programming.
- Hybrid MPI/OpenMP is recommended for many next generation architectures (Intel Xeon Phi for example), including NERSC-8 system, Cori.
- You should explore to add OpenMP now if your application is flat MPI only.

- 123 -

“OpenMP Basics and MPI/OpenMP Scaling”, Yun He, NERSC, 2015

The Big Picture

“OpenMP is fun” is not a sufficient justification for changing our programming model!

Threads and Library Interfaces

Attempt 1

Library spawns threads

```
void library_func(double *x, int N) {
  #pragma omp parallel for
  for (int i=0; i<N; ++i) x[i] = something_complicated();
}
```

Problems

Call from multi-threaded environment?

```
void user_func(double **y, int N) {
  #pragma omp parallel for
  for (int j=0; j<M; ++j) library_func(y[j], N);
}
```

Incompatible OpenMP runtimes (e.g. GCC vs. ICC)

Threads and Library Interfaces

Attempt 2

Use pthreads/TBB/etc. instead of OpenMP to spawn threads

Fixes incompatible OpenMP implementations (probably)

Problems

Still a problem with multi-threaded user environments

```
void user_func(double **y, int N) {
  #pragma omp parallel for
  for (int j=0; j<M; ++j) library_func(y[j], N);
}
```

Threads and Library Interfaces

Attempt 3

Hand back thread management to user

```
void library_func(ThreadInfo ti, double *x, int N) {
  int start = compute_start_index(ti, N);
  int stop  = compute_stop_index(ti, N);
  for (int i=start; i<stop; ++i)
    x[i] = something_complicated();
}
```

Implications

Users can use their favorite threading model

API requires one extra parameter

Extra boilerplate code required in user code

Threads and Library Interfaces

Reflection

Extra thread communication parameter

```
void library_func(ThreadInfo ti, double *x, int N) {...}
```

Rename thread management parameter

```
void library_func(Thread_Comm c, double *x, int N) {...}
```

Compare:

```
void library_func(MPI_Comm comm, double *x, int N) {...}
```

Conclusion

Prefer flat MPI over MPI+OpenMP for a composable software stack

MPI automatically brings better data locality