ICL Friday Seminar

September 14, 2012

Virtual Systolic Array for QR Decomposition

Jakub Kurzak
Piotr Luszczek
Mark Gates
Ichi Yamazaki
Yves Robert
Guillaume Aupy
Mathieu Faverge

Some Controversy

cool questions

- ✔ Can I run a 40x40 tiles matrix (1600 tiles) on ~1600 cores ?
- ✔ Can I run a 40K matrix on 40K cores ?
- ✔ Can I beat HPL (90% efficiency) with QR (2x operations) ?

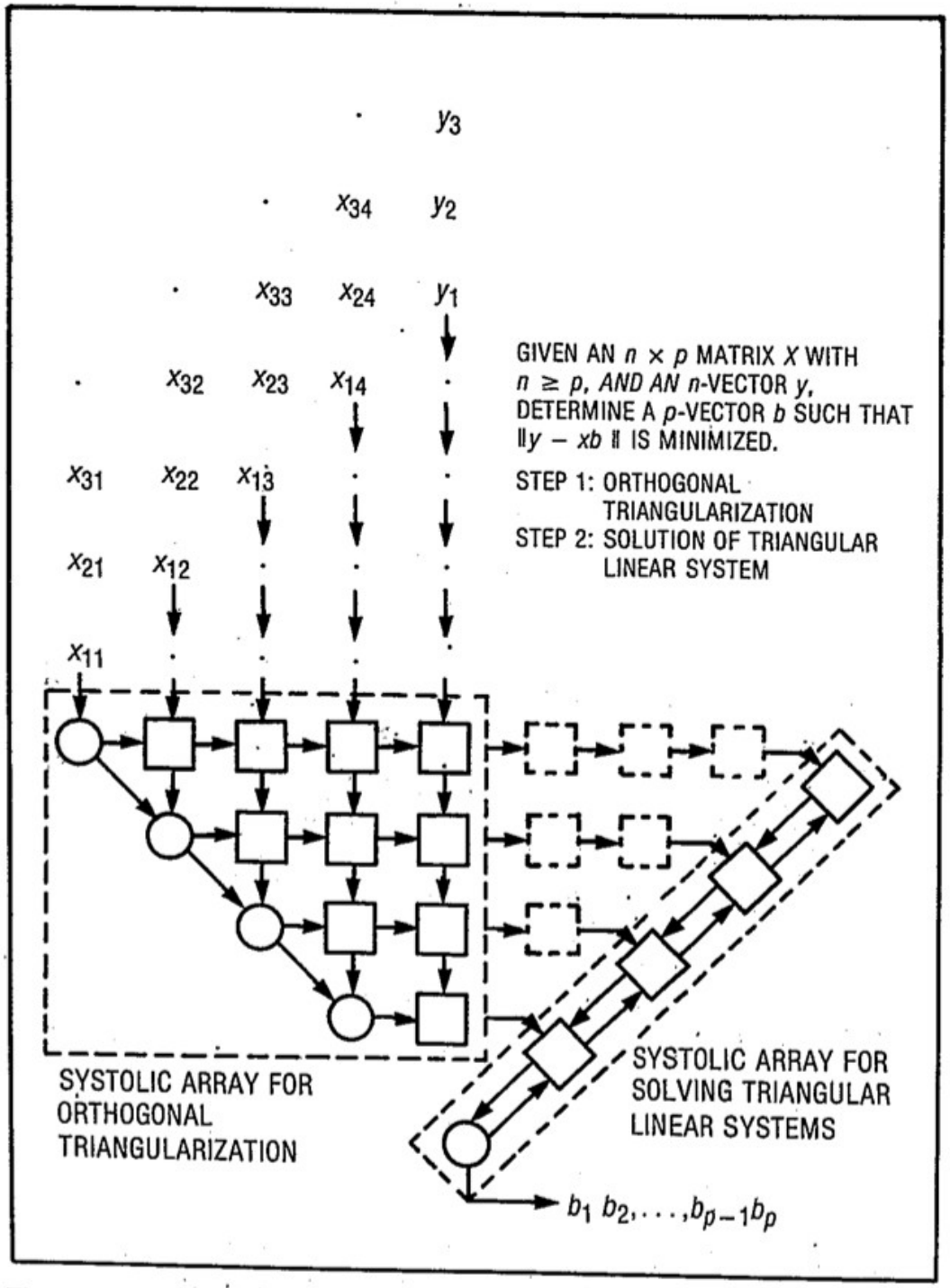


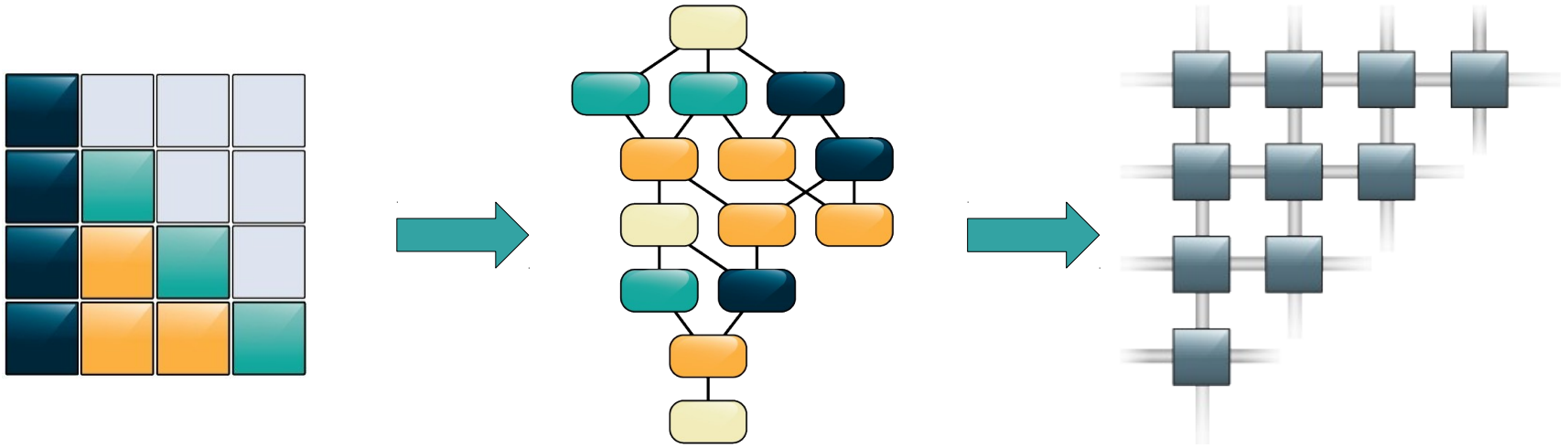
Figure 12. On-the-fly least-squares solutions using one- and two-dimensional systolic arrays, with $p = 4$.

parallel to the array presents no problems, even if the array is arbitrarily long. The systolic array (with data flowing in either one or opposite directions) will operate correctly despite the possibility of a large clock skew between its two ends.[30] However, large two-dimensional arrays may require slowdown of the global clock to compensate for clock skews. Except for this possible problem in the two-dimensional case, systolic designs are completely modular and expandable; they present no difficult synchronization or resource conflict problems. Software overhead associated with operations such as address indexing are totally eliminated in systolic systems. This advantage alone can mean a substantial performance improvement over conventional general-purpose computers. Simple, regular control and communication also imply simple, area-efficient layout or wiring—an important advantage in VLSI implementation.

In summary, systolic designs based on these criteria are *simple* (a consequence of properties 3 and 4), *modular* and *expandable* (property 4), and yield *high performance* (properties 1, 2, and 4). They therefore meet the architectural challenges for special-purpose systems. A unique characteristic of the systolic approach is that as the number of cells expands the system cost and performance increase proportionally, provided that the size of the underlying problem is sufficiently large. For example, a systolic convolution array can use an arbitrarily large number of cells cost-effectively, if the kernel size (that is, the number of weights) is large. This is in contrast to other parallel architectures which are seldom cost-effective for more than a small number of processors. From a user's point of view, a systolic system is easy to use—he simply pumps in the input data and then receives the results either on-the-fly or at the end of the computation.

PULSAR

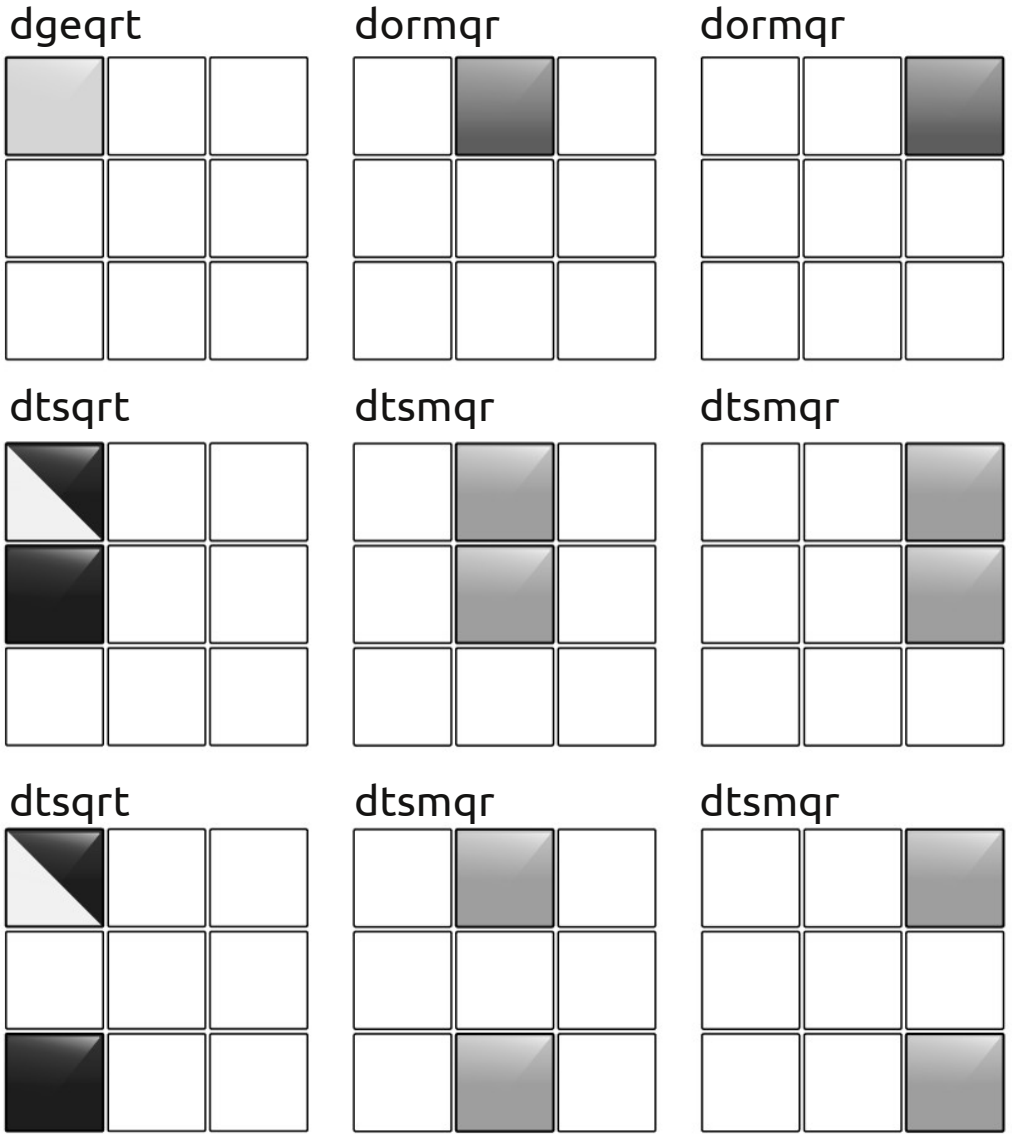
Parallel Unified Linear Algebra with Systolic ARrays



Virtualization

Tile QR

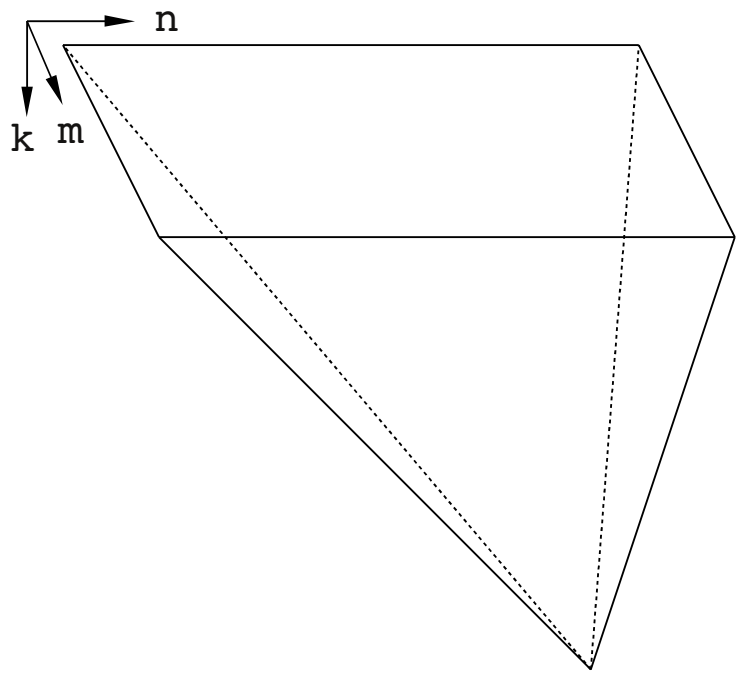
pLASMA kernels



Systolic QR

three nested loops = cone

```
FOR k=0 TO N-1
  DGEQRT(inout A_kk)
  FOR n=k+1 to N
    DORMQR(in A◣_kk, inout A_kn)
  FOR m=k+1 to N
    DTSQRT(inout A◣_kk, inout A_mk)
    FOR n=k+1 to N
      DTSMQR(in A_mk, inout A_kn, inout A_mn)
```



```
FOR k=0 TO N-1
  DGEQRT( inout A kk )
  FOR n=k+1 to N
    DORMQR( in A◹kk, inout A kn )
  FOR m=k+1 to N
    DTSQRT( inout A◹kk, inout A mk )
    FOR n=k+1 to N
      DTSMQR( in A mk, inout A kn, inout A mn )
```

DGEQRT_{kkk}

$1_{ARG} \leftarrow A_{k,k} \mid \mathrm{DTSMQR}_{k,k,k-1}$

$1_{ARG} \Rightarrow \mathrm{DORMQR}_{k,k+1..N,k}(\blacktriangleleft)$

$1_{ARG} \Rightarrow \mathrm{DTSQRT}_{k+1,k,k}(\blacktriangleleft)$

$1_{ARG} \Rightarrow A_{k,k}(\blacktriangleleft)$

DORMQR_{knk}

$1_{ARG} \leftarrow \mathrm{DGEQRT}_{k,k,k}(\blacktriangleleft)$

$2_{ARG} \leftarrow A_{k,n} \mid \mathrm{DTSMQR}_{k,n,k-1}$

$2_{ARG} \Rightarrow \mathrm{DTSMQR}_{k+1,n,k}$

$2_{ARG} \Rightarrow A_{k,n}$

DTSQRT_{mkk}

$1_{ARG} \leftarrow \mathrm{DGEQRT}_{m-1,k,k}(\blacktriangleleft) \mid \mathrm{DTSQRT}_{m-1,k,k}(\blacktriangleleft)$

$1_{ARG} \Rightarrow \mathrm{DTSQRT}_{m+1,k,k}(\blacktriangleleft) \mid A_{k,k}(\blacktriangleleft)$

$2_{ARG} \leftarrow A_{m,k} \mid \mathrm{DTSMQR}_{m,k,k-1}$

$2_{ARG} \Rightarrow \mathrm{DTSMQR}_{m,k+1..N,k}$

$2_{ARG} \Rightarrow A_{m,k}$

DTSMQR_{mnk}

$1_{ARG} \leftarrow \mathrm{DTSQRT}_{m,k,k}$

$2_{ARG} \leftarrow \mathrm{DORMQR}_{m-1,n,k} \mid \mathrm{DTSMQR}_{m-1,n,k}$

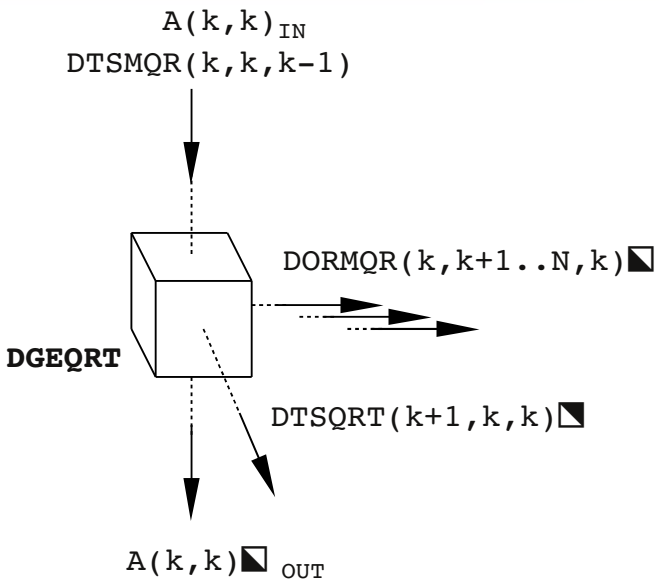
$2_{ARG} \Rightarrow \mathrm{DTSMQR}_{m+1,n,k} \mid A_{n,k}$

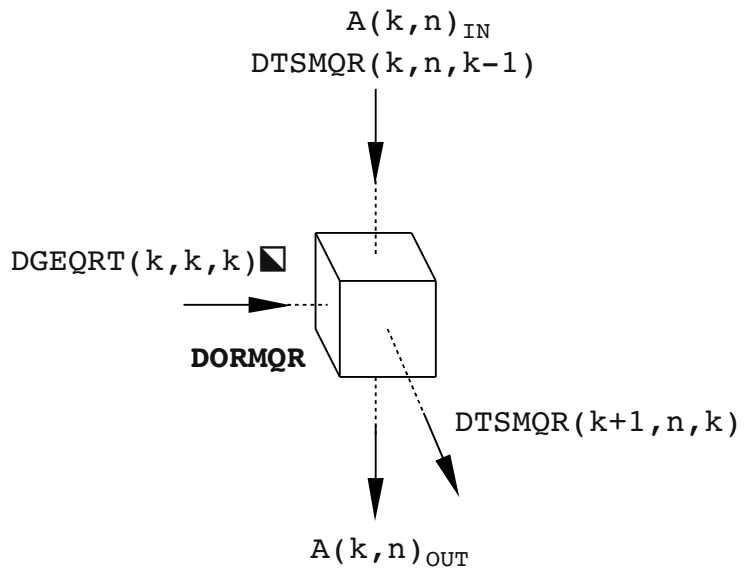
$3_{ARG} \leftarrow A_{m,n} \mid \mathrm{DTSMQR}_{m,n,k-1}$

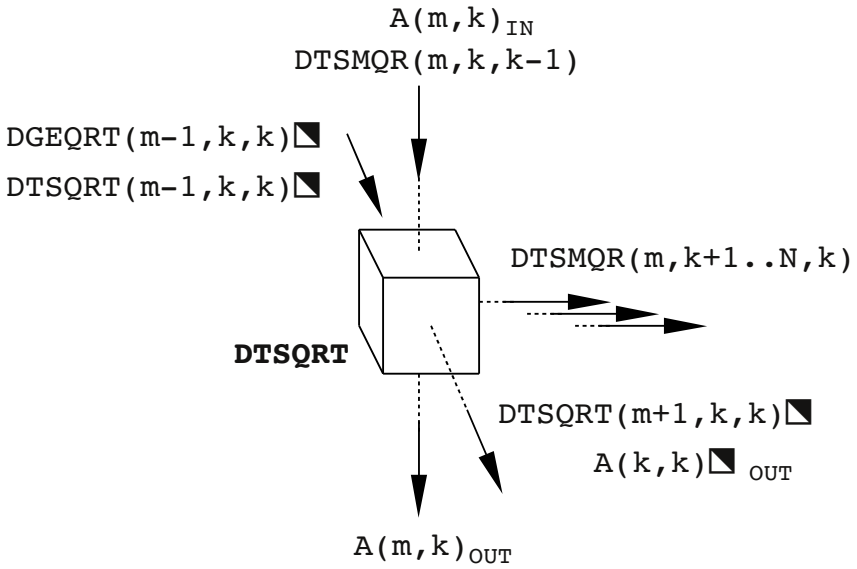
$3_{ARG} \Rightarrow \mathrm{DGEQRT}_{m,n,k+1} \mid \mathit{DORMQR}_{m,n,k+1} \mid$

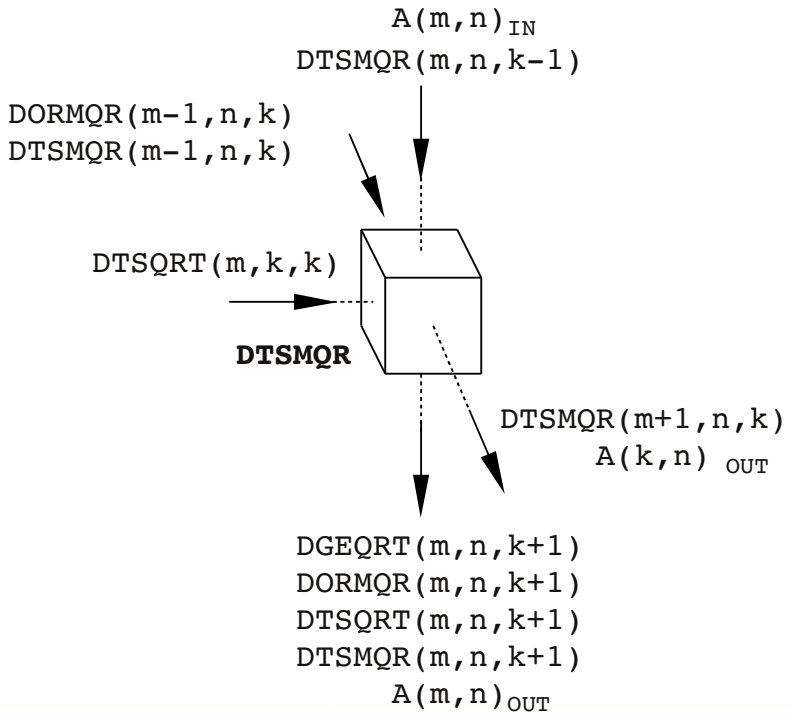
$\Rightarrow \mathrm{DTSQRT}_{m,n,k+1} \mid \mathit{DTSMQR}_{m,n,k+1} \mid$

$\Rightarrow A_{m,n}$


A(k,k) IN
DTSMQR(k,k,k-1)
DORMQR(k,k+1..N,k)
DGEQRT
DTSQRT(k+1,k,k)
A(k,k) OUT

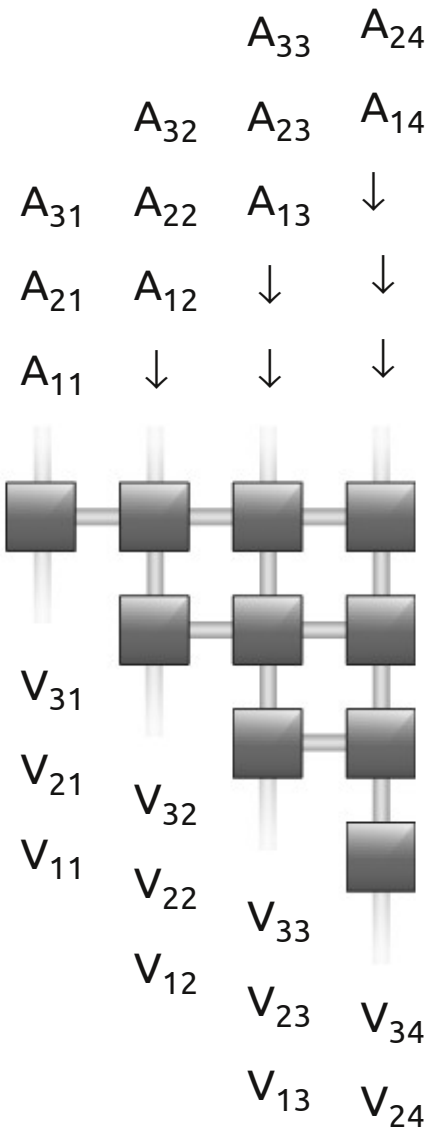

A(k,n) IN
DTSMQR(k,n,k-1)
DGEQRT(k,k,k)
DORMQR
DTSMQR(k+1,n,k)
A(k,n) OUT


A(m,k) IN
DTSMQR(m,k,k-1)
DGEQRT(m-1,k,k)
DTSQRT(m-1,k,k)
DTSMQR(m,k+1..N,k)
DTSQRT
DTSQRT(m+1,k,k)
A(k,k) OUT
A(m,k) OUT


A(m,n) IN
DTSMQR(m,n,k-1)
DORMQR(m-1,n,k)
DTSMQR(m-1,n,k)
DTSQRT(m,k,k)
DTSMQR
DTSMQR(m+1,n,k)
A(k,n) OUT
DGEQRT(m,n,k+1)
DORMQR(m,n,k+1)
DTSQRT(m,n,k+1)
DTSMQR(m,n,k+1)
A(m,n) OUT

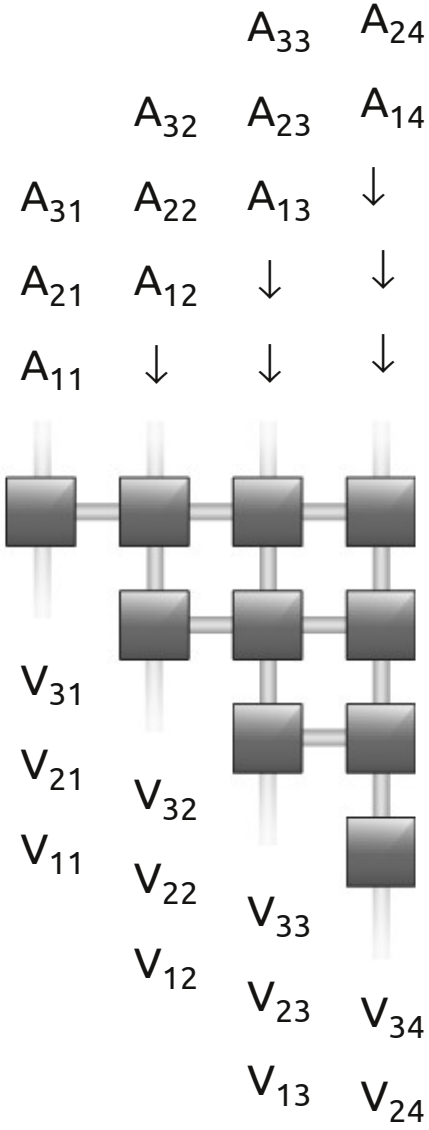
Systolic QR

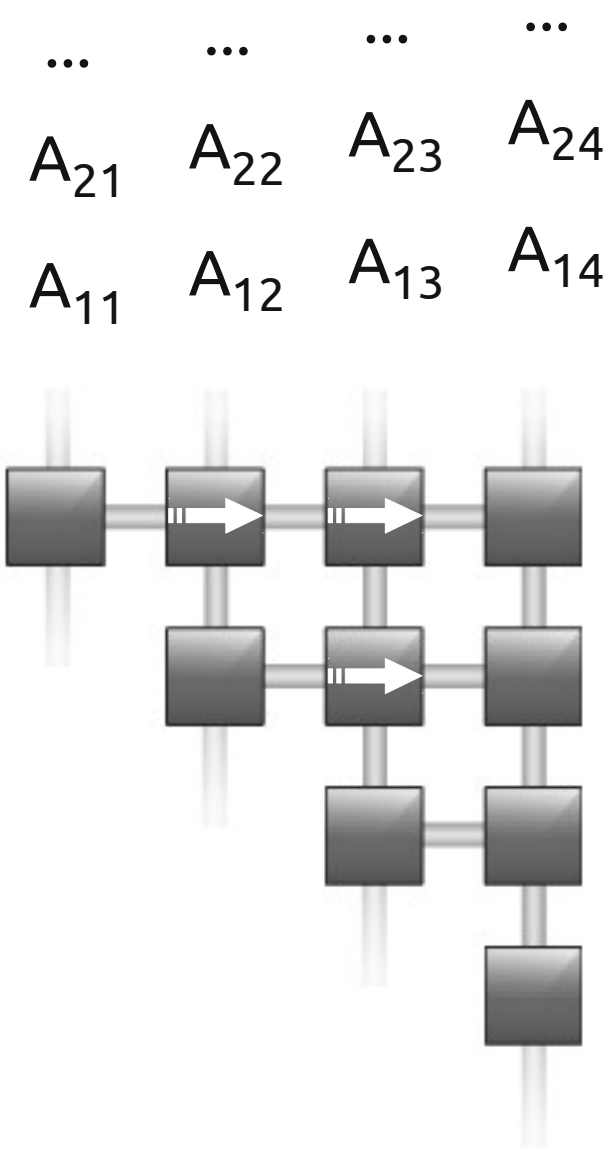
systolic array for tile QR



Systolic QR

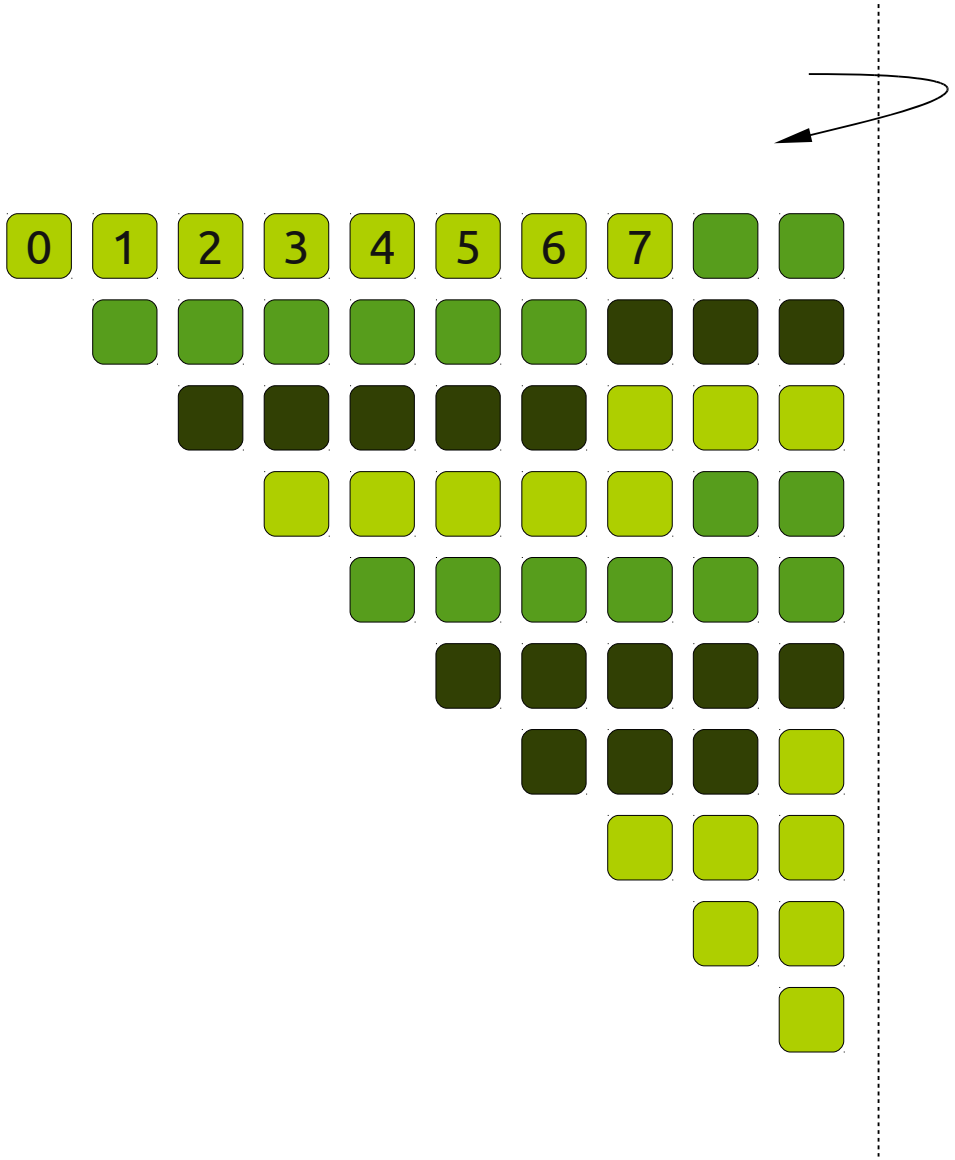
data bypass / quick forwarding





Virtualization

mapping of systolic processing units to cores



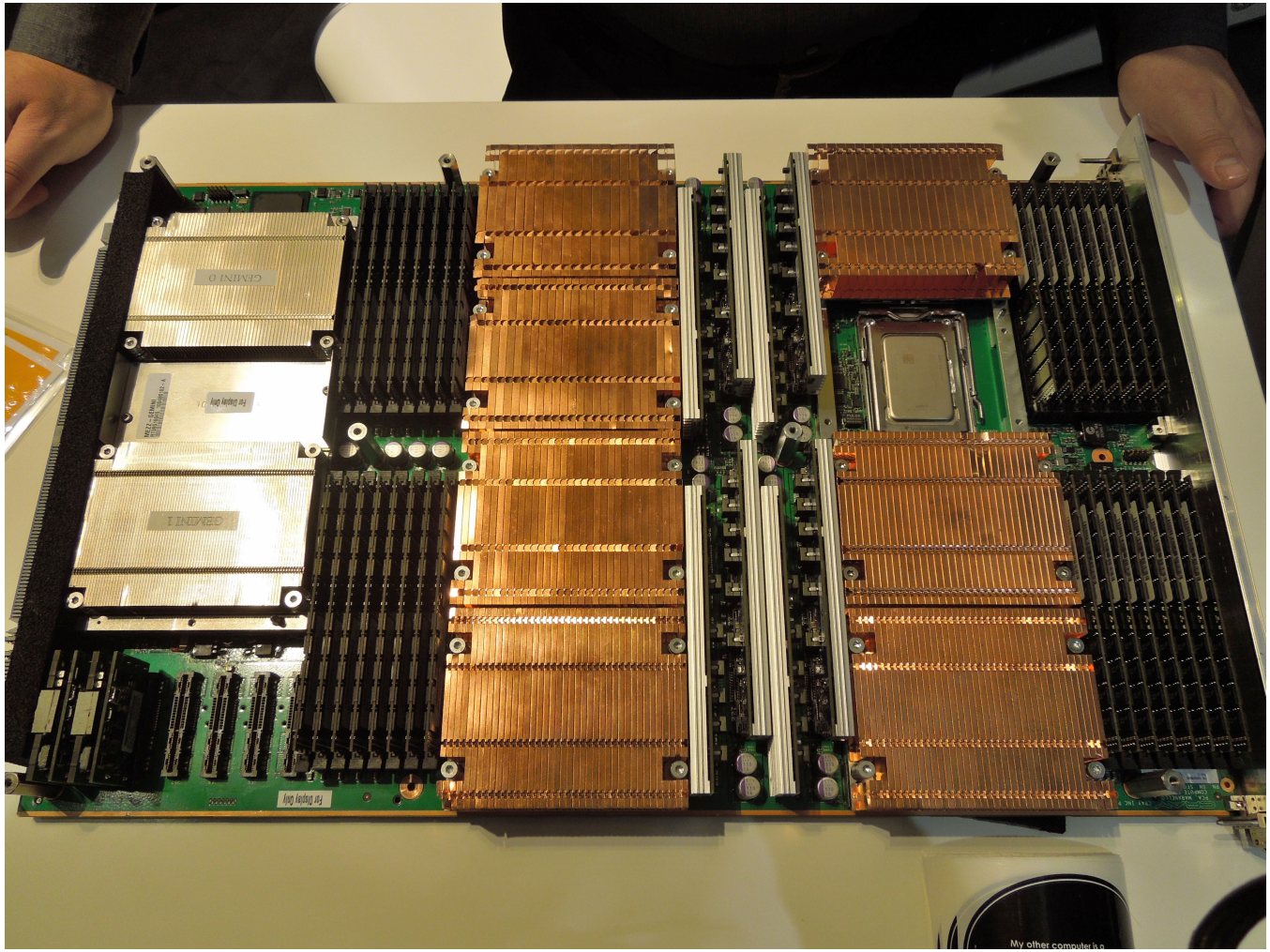
Kraken

National Institute for Computational Science

- ✔ two 2.6 GHz six-core AMD Opterons
- ✔ 16 GB memory
- ✔ Cray SeaStar2+ Interconnects
- ✔ ca. 10K nodes (100K cores)

Kraken

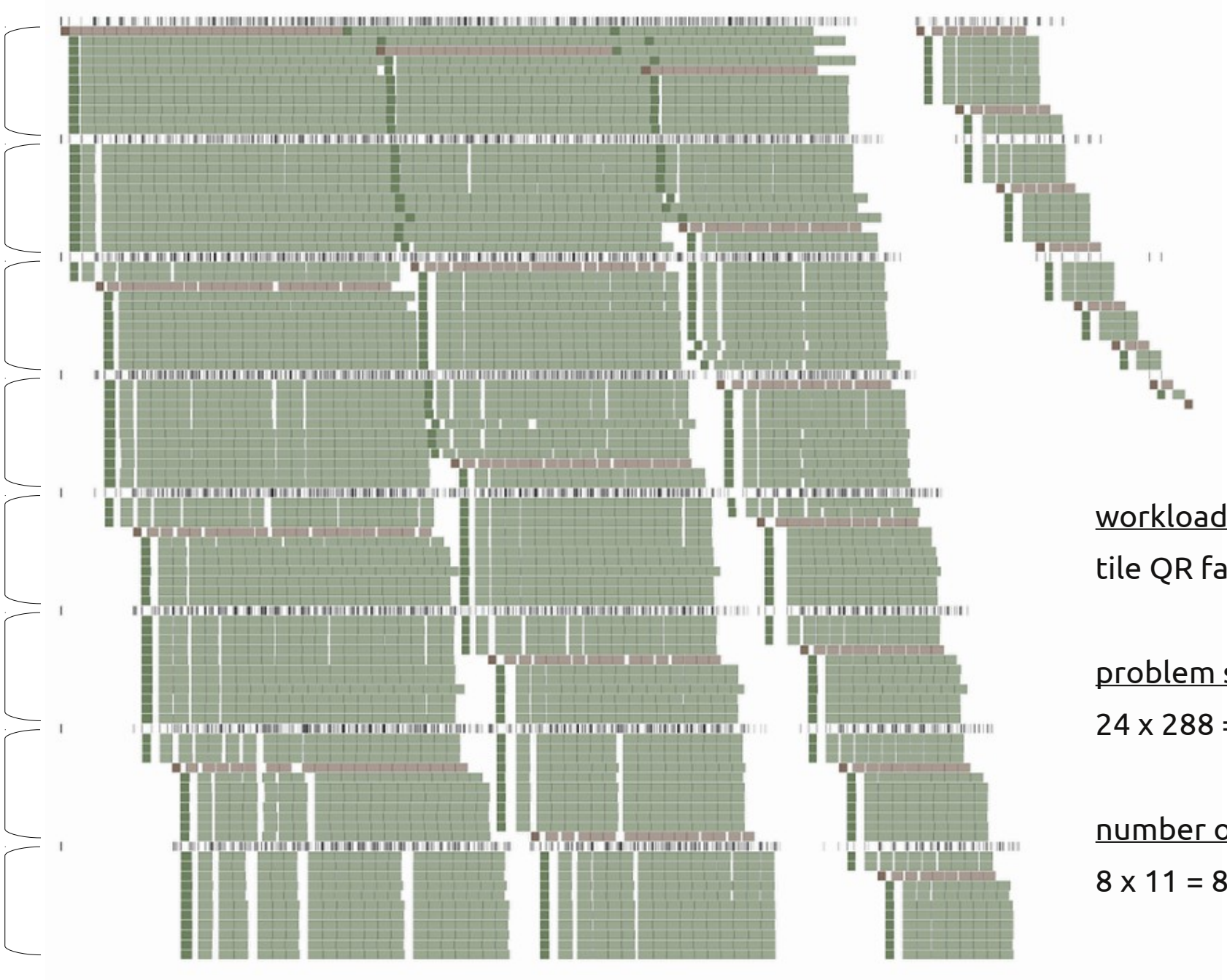
National Institute for Computational Science



Implementation

multicore + MPI

- ✔ pthreads within the node
- ✔ MPI between nodes
- ✔ flat communicatiom model – core to core
- ✔ communication proxy – dedicated core



workload

tile QR factorization

problem size

24 x 288 = 6,912

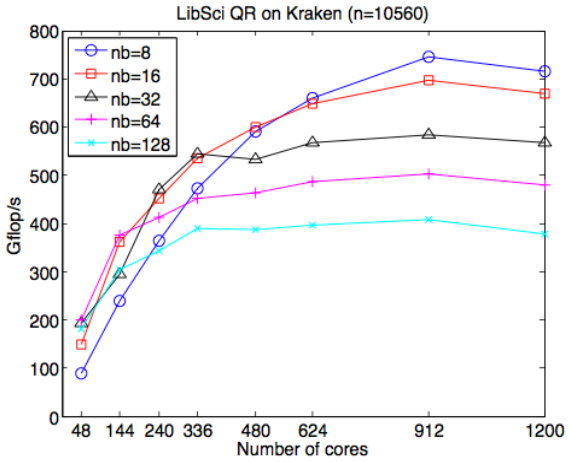
number of cores

8 x 11 = 88 cores

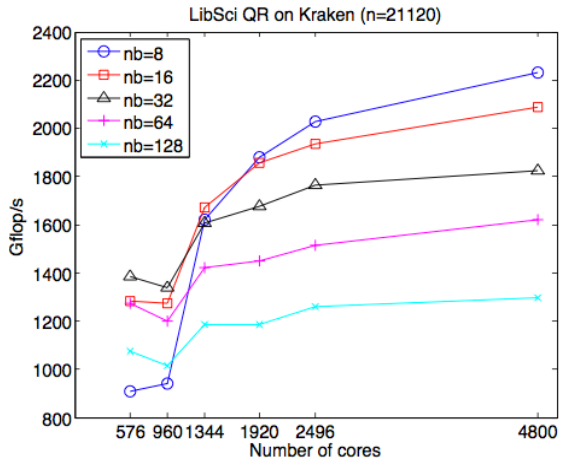
Cray LibSci

tuning

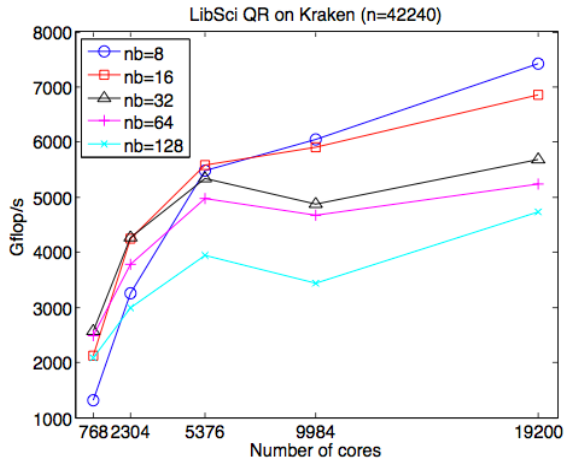
10K x 10K



20K x 20K

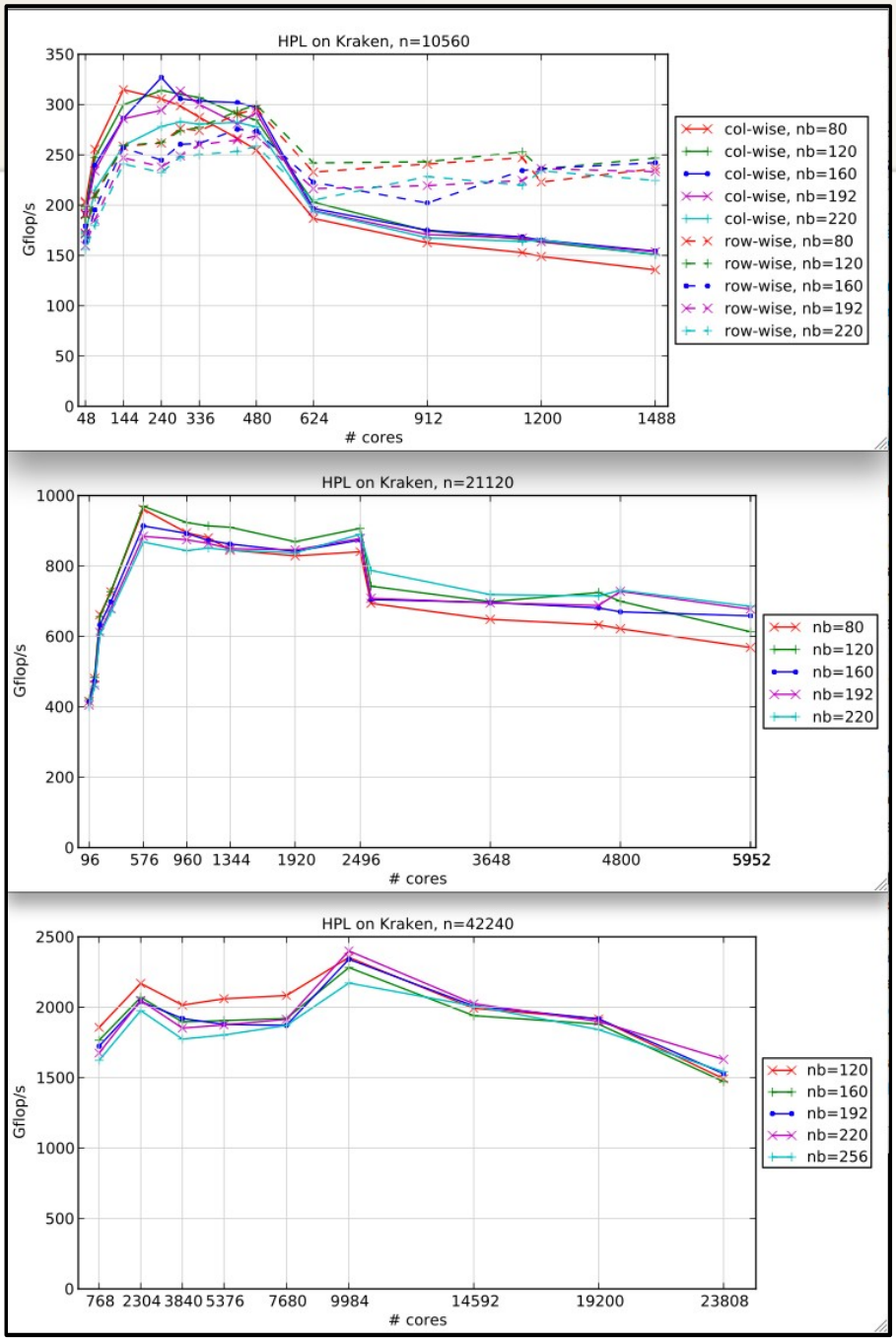


40K x 40K



Netlib HPL

tuning



10K x 10K

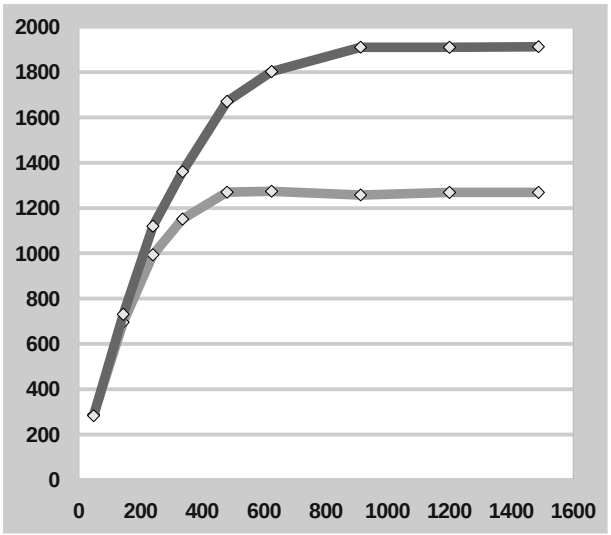
20K x 20K

40K x 40K

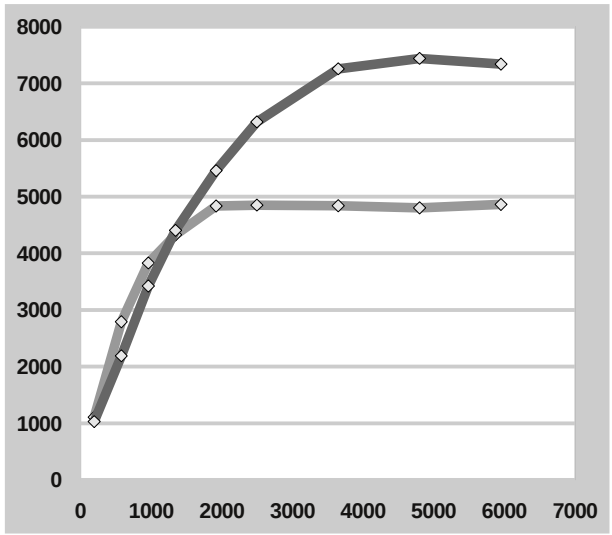
PULSAR QR

Performance

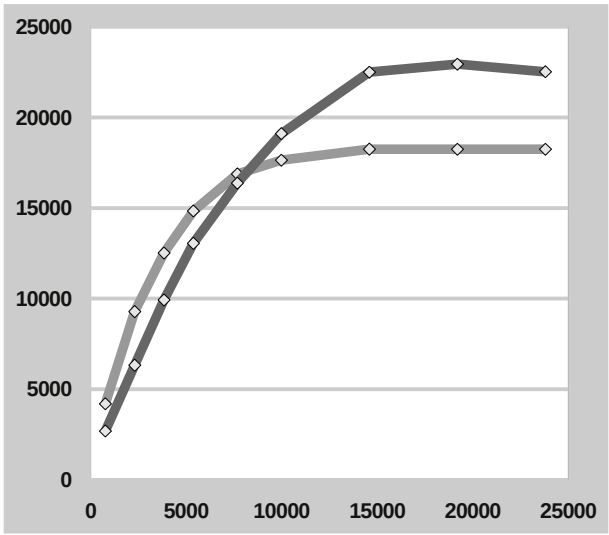
10K x 10K



20K x 20K



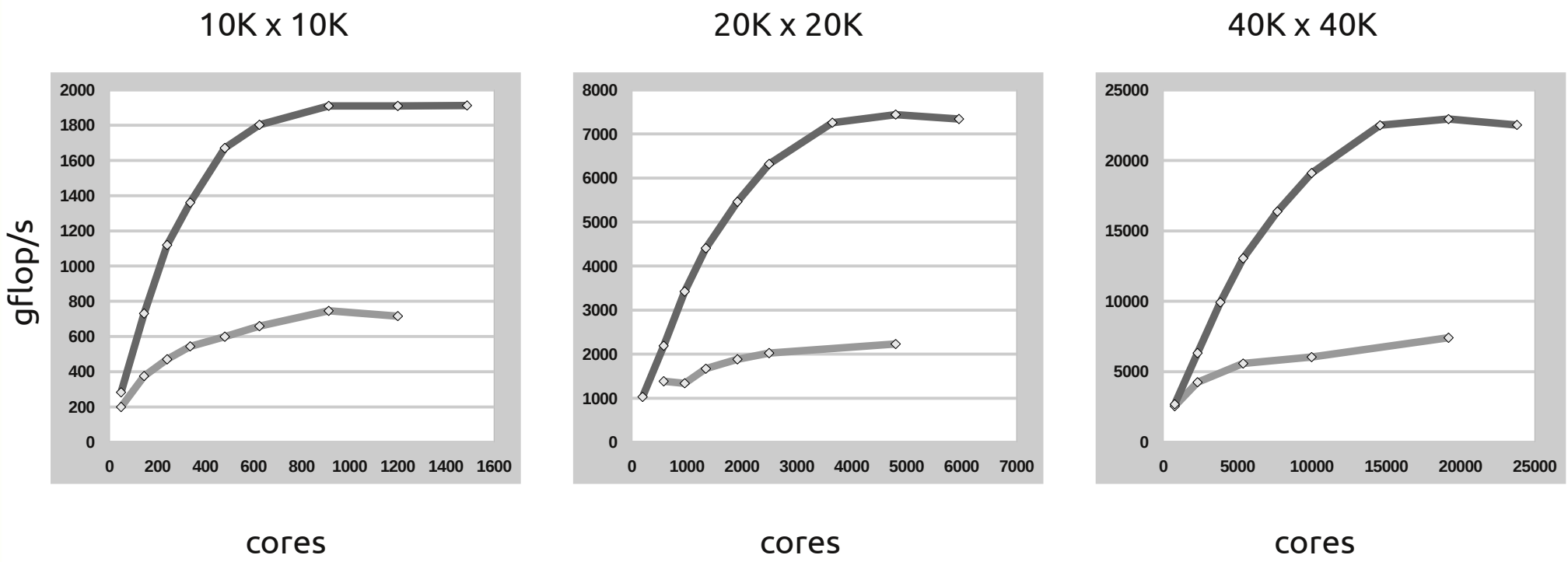
40K x 40K



IB = 64
NB = 192, 256

PULSAR QR

Performance vs. LibSci

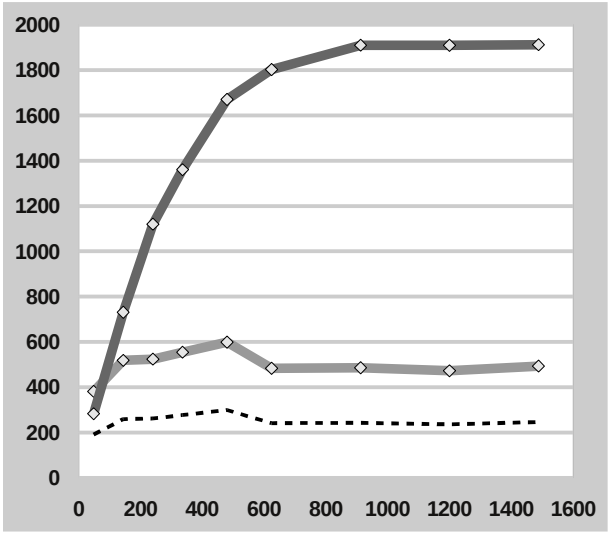


IB = 64
NB = 192

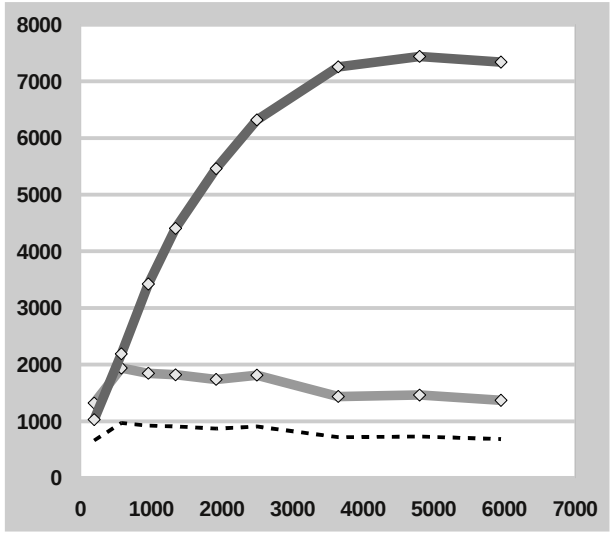
PULSAR QR

Performance vs. HPL

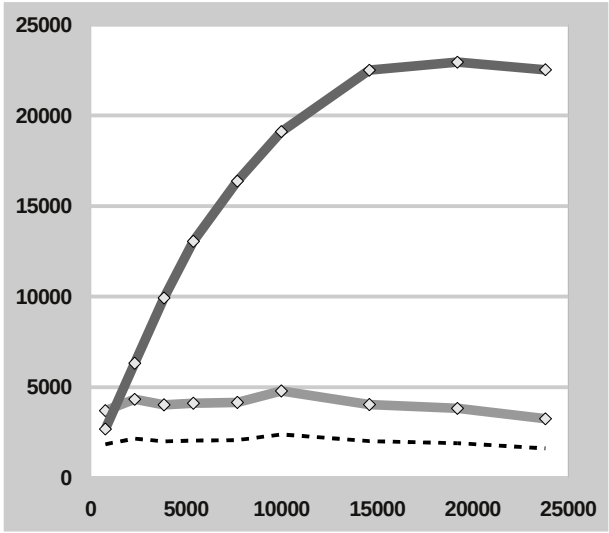
10K x 10K



20K x 20K

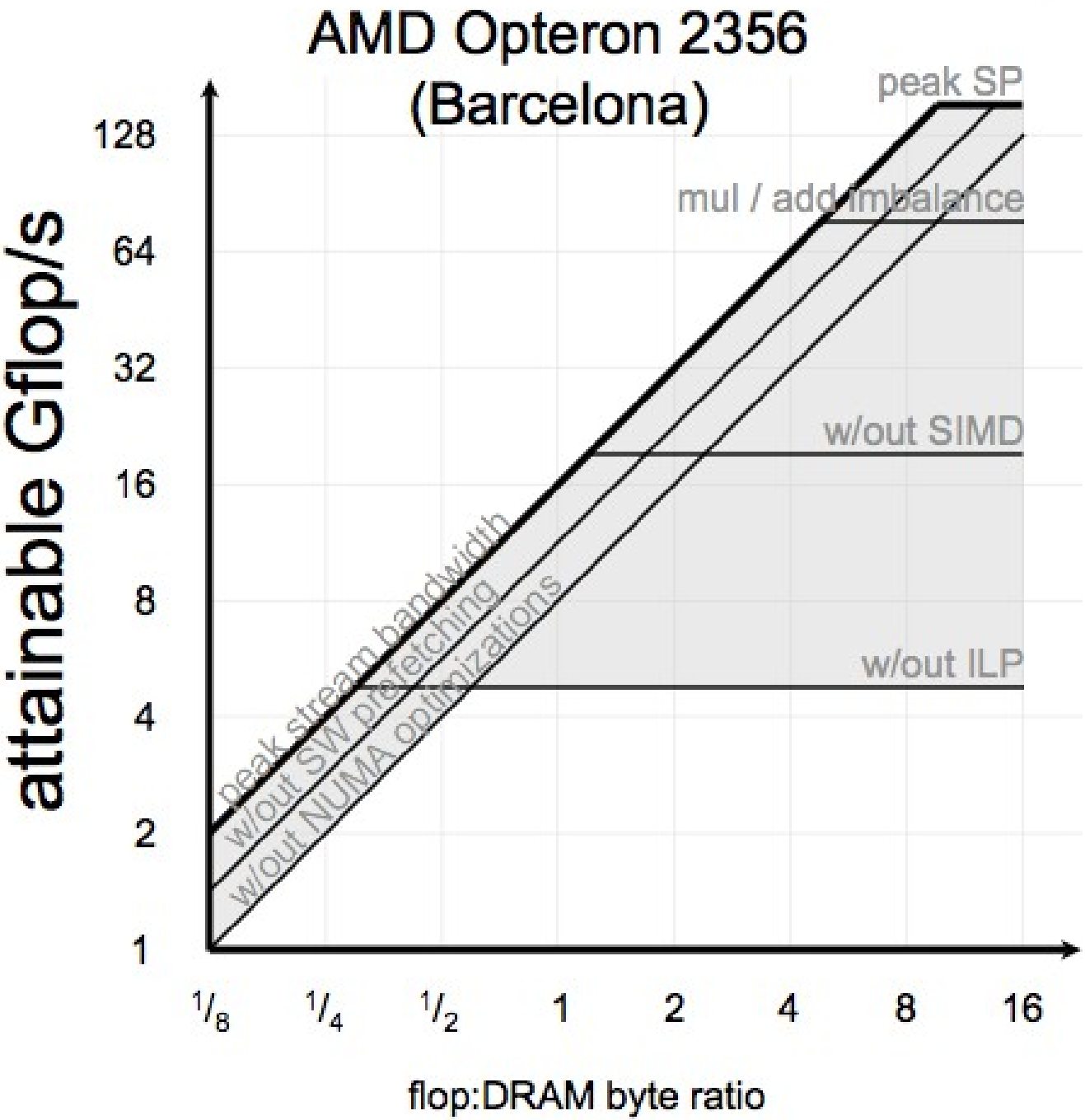


40K x 40K



IB = 64
NB = 192

Roofline model for Opteron
(powerpoint doodle)

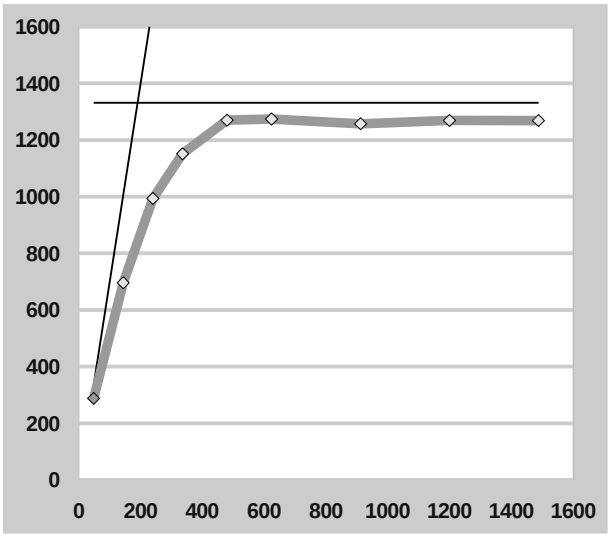


- Final Roofline

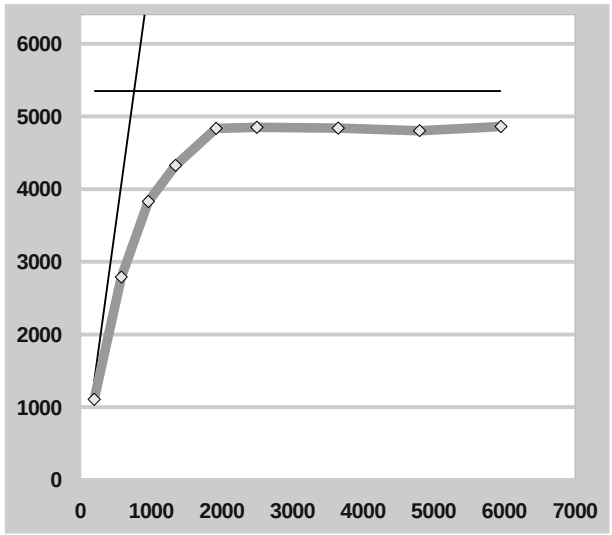
PULSAR QR

Parallel Roofline Model

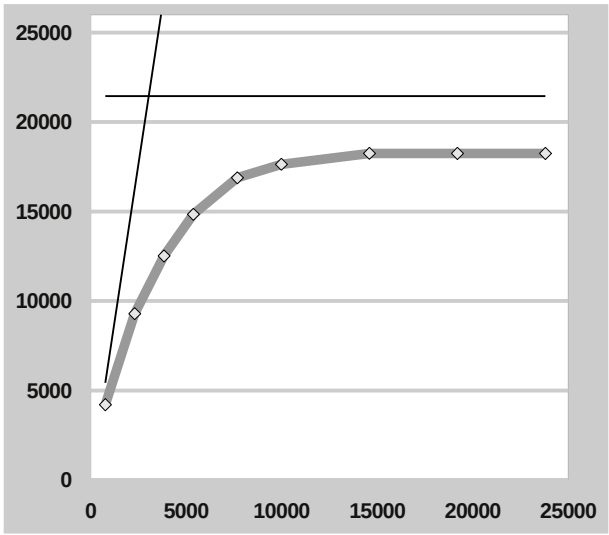
10K x 10K



20K x 20K



40K x 40K

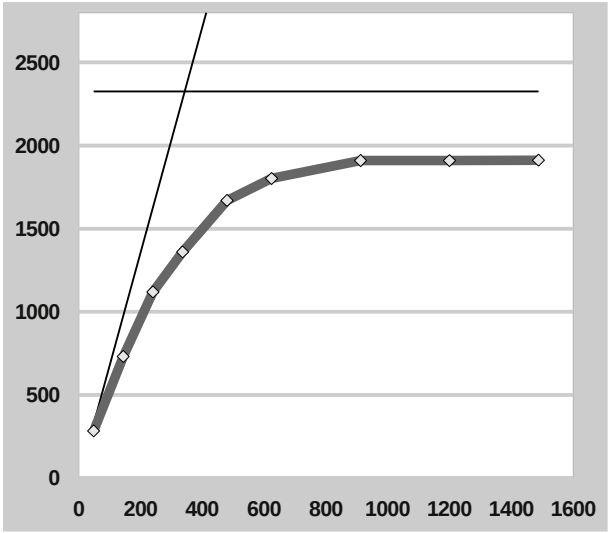


IB = 64
NB = 256

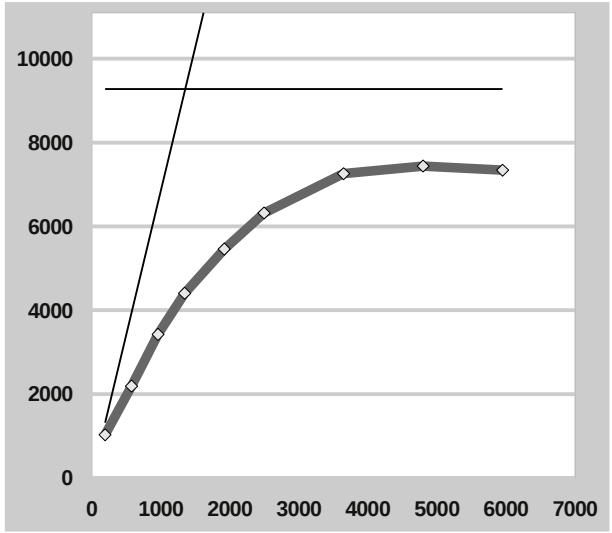
PULSAR QR

Parallel Roofline Model

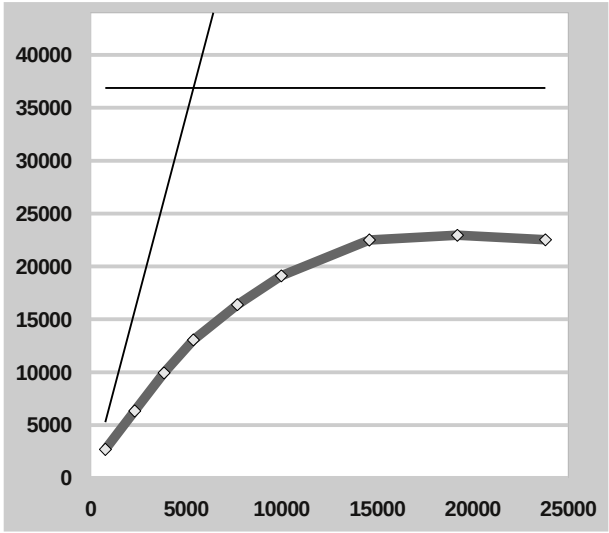
10K x 10K



20K x 20K



40K x 40K



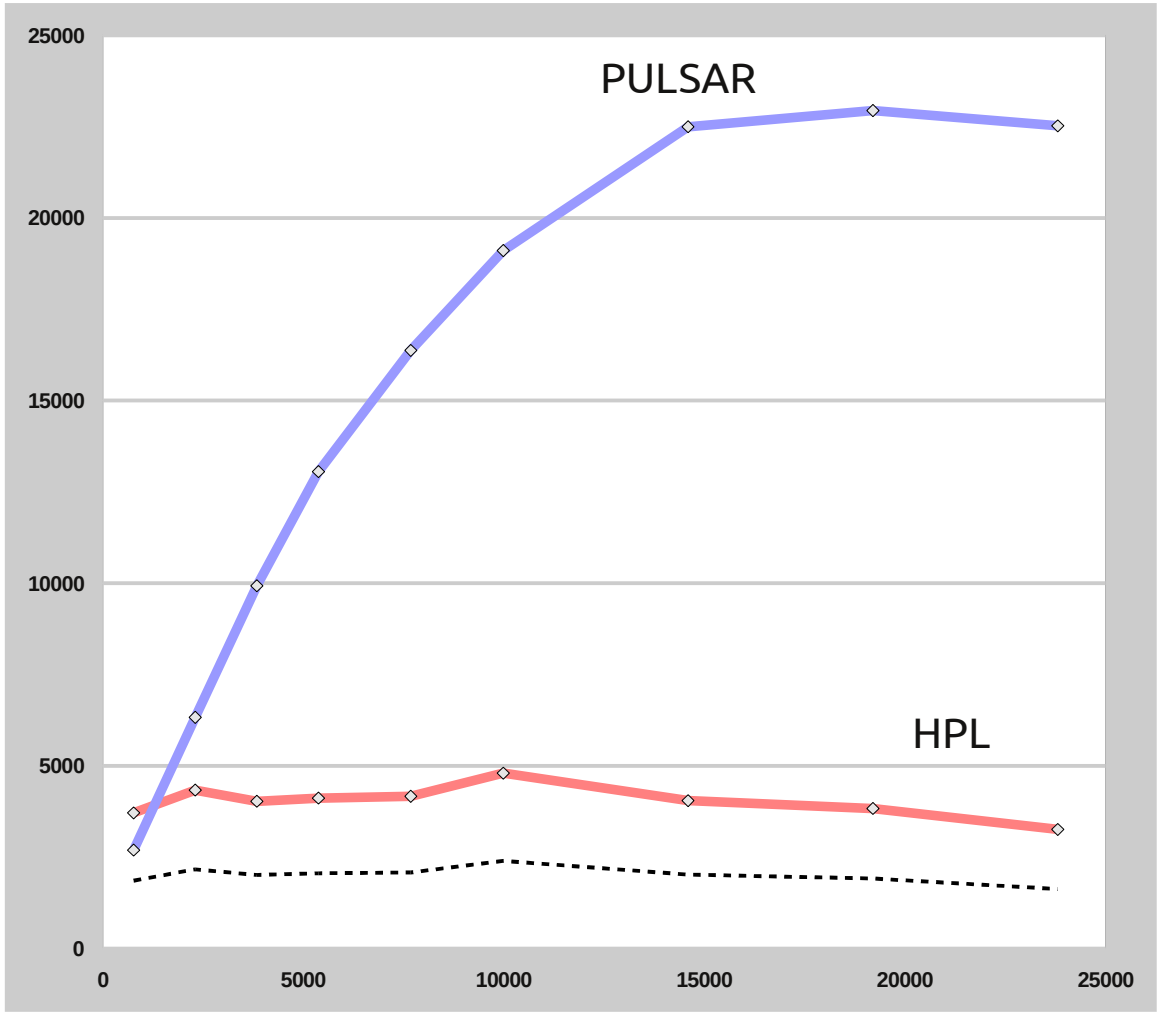
IB = 64
NB = 192

Summary
cool observations

- ✔ running 40x40 matrix (1600 tiles) on ~1600 cores
- ✔ running 40K matrix on 20K cores
- ✔ using 2000 nodes for a problem that fits on one node
- ✔ 7 times faster than HPL
- ✔ roofline accurate to within 5%

Questions
and answers

COMMUNICATE !



~40K size
~200 tiles

IB = 64
NB = 192