Paralelní a distribuované výpočty (B4B36PDV)

Jakub Mareček, Michal Jakob

jakub.marecek@fel.cvut.cz

Artificial Intelligence Center
Department of Computer Science
Faculty of Electrical Engineering
Czech Technical University in Prague

Dnes s využitím materiálů Brana Bošanského: https://cw.fel.cvut.cz/b192/courses/b4b36pdv/lectures/start

Dnešní přednáška

Techniky paralelizace 3

Chci paralelizovat učení neuronky

Jak na to?

Dnešní přednáška

Techniky paralelizace 3

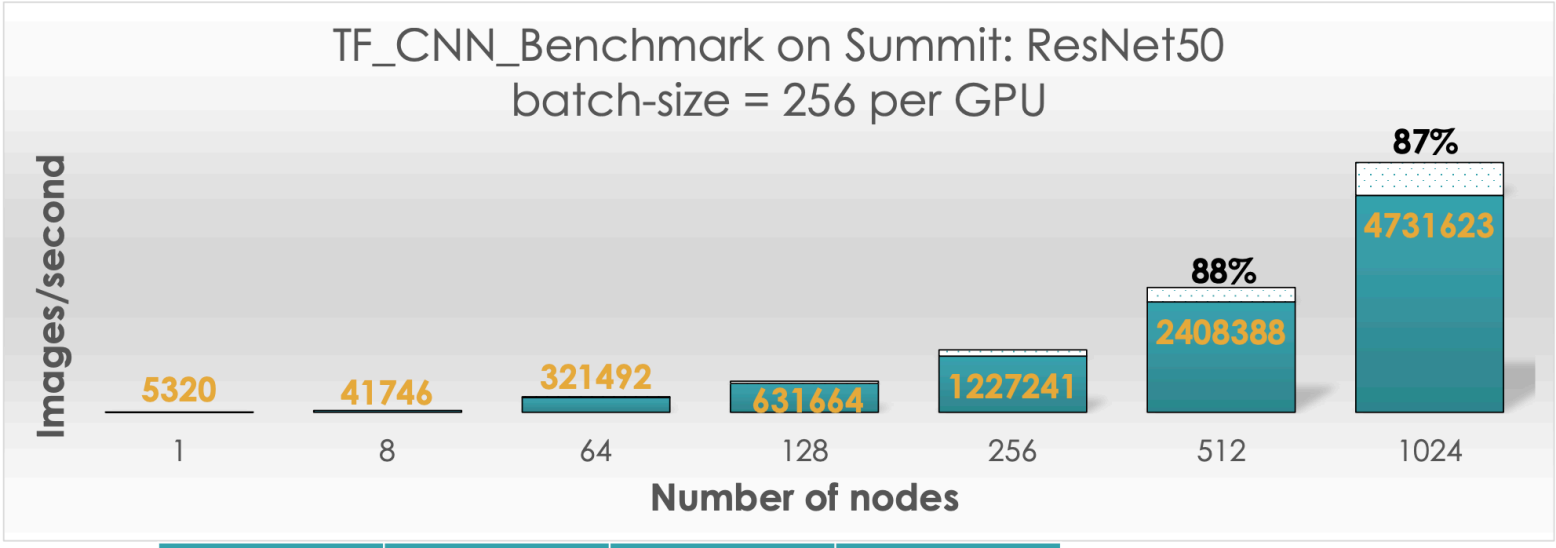
Chci paralelizovat maticový algoritmus

Jak na to?

Demotivace

Prototypování numerických algoritmů

Performance baselines: ResNet50 on ImageNet



Nodes	Mini-batch size	Top-1 Val accuracy	Training time (min)
16	12288	0.750	27
32	12288	0.766	17
64	15360	0.763	12

TF.distribute & Horovod + LARS

code: TensorFlow distributed example

https://www.olcf.ornl.gov/wp-content/uploads/2020/02/MLDL-on-Summit-June2020.pdf

Motivace

Prototypování numerických algoritmů

```
#include <iostream>
#include <vector>
#include "mkldnn.hpp"

using namespace mkldnn;

…

int main(int argc, char **argv) {
    try {
        simple_net();
        std::cout << "ok\n";
    } catch (error &e) {
        std::cerr << "status: " << e.status << std::endl;
        std::cerr << "message: " << e.message << std::endl;
    }
    return 0;
}
```

https://oneapi-src.github.io/oneDNN/v1.0/cpu_rnn_inference_int8_cpp.html

Motivace

Prototypování numerických algoritmů

```
#include <iostream>
#include <vector>
#include "mkldnn.hpp"

using namespace mkldnn;
using dim_t = mkldnn::memory::dim;

int main(int argc, char **argv) {
    try {
        const dim_t n = 64;
        // column-major order (sloupce souvisle)
        std::vector<float> A(n*n, 1.0f);
        std::vector<float> B(n*n, 1.0f);
        std::vector<float> C(n*n, 1.0f);
        // https://oneapi-src.github.io/oneDNN/v0/group__c__api__blas.html
        mkldnn_status_t status = mkldnn_sgemm('N', 'N', n, n, n,
                    1.f, A.data(), n, B.data(), n,
                    0.f, C.data(), n);
        std::cerr << "status: " << status << std::endl;
    } catch (error &e) {
        std::cerr << "status: " << e.status << std::endl;
        std::cerr << "message: " << e.message << std::endl;
    }
    return 0;
}
```

Motivace

Prototypování numerických algoritmů

Developer Reference for Intel® oneAPI Math Kernel Library - C

Developer Reference

Version: 2021.2 Last Updated: 03/26/2021 Public Content Download as PDF

Search this document

cblas_?gemv

Computes a matrix-vector product using a general matrix.

Syntax

```
void cblas_sgemv (const CBLAS_LAYOUT Layoutconst CBLAS_TRANSPOSE transconst MKL_INT mconst MKL_INT nconst
float alphaconst float *aconst MKL_INT ldaconst float *xconst MKL_INT incxconst float betafloat *yconst
MKL_INT incy);

void cblas_dgemv (const CBLAS_LAYOUT Layoutconst CBLAS_TRANSPOSE transconst MKL_INT mconst MKL_INT nconst
double alphaconst double *aconst MKL_INT ldaconst double *xconst MKL_INT incxconst double betadouble
*yconst MKL_INT incy);

void cblas_cgemv (const CBLAS_LAYOUT Layoutconst CBLAS_TRANSPOSE transconst MKL_INT mconst MKL_INT nconst
void *alphaconst void *aconst MKL_INT ldaconst void *xconst MKL_INT incxconst void *betavoid *yconst
MKL_INT incy);
```

Motivace

Prototypování numerických algoritmů

Level 1 BLAS

dim scalar vector vector scalars	5-element array		prefixes
SUBROUTINE xROTG (A, B, C, S)		Generate plane rotation	S, D
SUBROUTINE xROTMG(D1, D2, A, B,	PARAM)	Generate modified plane rotation	S, D
SUBROUTINE xROT (N, X, INCX, Y, INCY, C, S)		Apply plane rotation	S, D
SUBROUTINE xROTM (N, X, INCX, Y, INCY,	PARAM)	Apply modified plane rotation	S, D
SUBROUTINE xSWAP (N, X, INCX, Y, INCY)		$x \leftrightarrow y$	S, D, C, Z
SUBROUTINE xSCAL (N, ALPHA, X, INCX)		$x \leftarrow \alpha x$	S, D, C, Z, CS, ZD
SUBROUTINE xCOPY (N, X, INCX, Y, INCY)		$y \leftarrow x$	S, D, C, Z
SUBROUTINE xAXPY (N, ALPHA, X, INCX, Y, INCY)		$y \leftarrow \alpha x + y$	S, D, C, Z
FUNCTION xDOT (N, X, INCX, Y, INCY)		$dot \leftarrow x^T y$	S, D, DS
FUNCTION xDOTU (N, X, INCX, Y, INCY)		$dot \leftarrow x^T y$	C, Z
FUNCTION xDOTC (N, X, INCX, Y, INCY)		$dot \leftarrow x^H y$	C, Z
FUNCTION xxDOT (N, X, INCX, Y, INCY)		$dot \leftarrow \alpha + x^T y$	SDS
FUNCTION xNRM2 (N, X, INCX)		$nrm2 \leftarrow \|\|x\|\|_2$	S, D, SC, DZ
FUNCTION xASUM (N, X, INCX)		$asum \leftarrow \|\|re(x)\|\|_1 + \|\|im(x)\|\|_1$	S, D, SC, DZ
FUNCTION IxAMAX(N, X, INCX)		$amax \leftarrow 1^{st} k \ni \|re(x_k)\| + \|im(x_k)\| = max(\|re(x_i)\| + \|im(x_i)\|)$	S, D, C, Z

Level 2 BLAS

options dim b-width scalar matrix vector scalar vector			
xGEMV (TRANS, M, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)	$y \leftarrow \alpha Ax + \beta y, y \leftarrow \alpha A^T x + \beta y, y \leftarrow \alpha A^H x + \beta y, A - m \times n$	S, D, C, Z
xGBMV (TRANS, M, N, KL, KU, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)	$y \leftarrow \alpha Ax + \beta y, y \leftarrow \alpha A^T x + \beta y, y \leftarrow \alpha A^H x + \beta y, A - m \times n$	S, D, C, Z
xHEMV (UPLO, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	C, Z
xHBMV (UPLO, N, K, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	C, Z
xHPMV (UPLO, N, ALPHA, AP, X, INCX, BETA, Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	C, Z
xSYMV (UPLO, N, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	S, D
xSBMV (UPLO, N, K, ALPHA, A, LDA, X, INCX, BETA, Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	S, D
xSPMV (UPLO, N, ALPHA, AP, X, INCX, BETA, Y, INCY)	$y \leftarrow \alpha Ax + \beta y$	S, D
xTRMV (UPLO, TRANS, DIAG, N, A, LDA, X, INCX)	$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$	S, D, C, Z
xTBMV (UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX)	$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$	S, D, C, Z
xTPMV (UPLO, TRANS, DIAG, N, AP, X, INCX)	$x \leftarrow Ax, x \leftarrow A^T x, x \leftarrow A^H x$	S, D, C, Z
xTRSV (UPLO, TRANS, DIAG, N, A, LDA, X, INCX)	$x \leftarrow A^{-1}x, x \leftarrow A^{-T}x, x \leftarrow A^{-H}x$	S, D, C, Z
xTBSV (UPLO, TRANS, DIAG, N, K, A, LDA, X, INCX)	$x \leftarrow A^{-1}x, x \leftarrow A^{-T}x, x \leftarrow A^{-H}x$	S, D, C, Z
xTPSV (UPLO, TRANS, DIAG, N, AP, X, INCX)	$x \leftarrow A^{-1}x, x \leftarrow A^{-T}x, x \leftarrow A^{-H}x$	S, D, C, Z
options dim scalar vector vector matrix		
xGER (M, N, ALPHA, X, INCX, Y, INCY, A, LDA)	$A \leftarrow \alpha xy^T + A, A - m \times n$	S, D
xGERU (M, N, ALPHA, X, INCX, Y, INCY, A, LDA)	$A \leftarrow \alpha xy^T + A, A - m \times n$	C, Z
xGERC (M, N, ALPHA, X, INCX, Y, INCY, A, LDA)	$A \leftarrow \alpha xy^H + A, A - m \times n$	C, Z
xHER (UPLO, N, ALPHA, X, INCX, A, LDA)	$A \leftarrow \alpha xx^H + A$	C, Z
xHPR (UPLO, N, ALPHA, X, INCX, AP)	$A \leftarrow \alpha xx^H + A$	C, Z
xHER2 (UPLO, N, ALPHA, X, INCX, Y, INCY, A, LDA)	$A \leftarrow \alpha xy^H + y(\alpha x)^H + A$	C, Z
xHPR2 (UPLO, N, ALPHA, X, INCX, Y, INCY, AP)	$A \leftarrow \alpha xy^H + y(\alpha x)^H + A$	C, Z
xSYR (UPLO, N, ALPHA, X, INCX, A, LDA)	$A \leftarrow \alpha xx^T + A$	S, D
xSPR (UPLO, N, ALPHA, X, INCX, AP)	$A \leftarrow \alpha xx^T + A$	S, D
xSYR2 (UPLO, N, ALPHA, X, INCX, Y, INCY, A, LDA)	$A \leftarrow \alpha xy^T + \alpha yx^T + A$	S, D
xSPR2 (UPLO, N, ALPHA, X, INCX, Y, INCY, AP)	$A \leftarrow \alpha xy^T + \alpha yx^T + A$	S, D

http://www.netlib.org/blas/blasqr.pdf

Dnešní přednáška

Techniky paralelizace 3

Chci paralelizovat maticový algoritmus

Jak na to?

Maticové operace

Násobení matice vektorem

a11	a12	a13	a14	a15		x1		y1
a21	…					x2		y2
…					X	x3	=	y3
						x4		y4
				a55		x5		y5

$$y_1 = \sum_{\{i=1,\ldots,5\}} a_{1i} \,.\, x_i$$

Maticové operace

Násobení matice vektorem

- Jak paralelizovat?

a11	a12	a13	a14	a15		x1		y1
a21	…					x2		y2
…					X	x3	=	y3
						x4		y4
				a55		x5		y5

$$y_1 = \sum_{\{i=1,\ldots,5\}} a_{1i} \,.\, x_i$$

Maticové operace

Násobení matice vektorem

- Zřejmá paralelizace – výpočet každé složky y je nezávislý a všechny mohou být spočteny paralelně

P1 →	a11	a12	a13	a14	a15		x1		y1
P2 →	a21	…					x2		y2
	…					X	x3	=	y3
…							x4		y4
					a55		x5		y5

$$y_1 = \sum_{\{i=1,\ldots,5\}} a_{1i} \,.\, x_i$$

Maticové operace

Násobení matice vektorem

- Zřejmá paralelizace – výpočet každé složky y je nezávislý a všechny mohou být spočteny paralelně

```
void multiply(std::vector<int>& A, std::vector<int>& x, std::vector<int>& y) {
#pragma omp declare reduction(vec_int_plus : std::vector<int> : \
                std::transform(omp_out.begin(), omp_out.end(), omp_in.begin(), omp_out.begin(), std::plus<int>())) \
                initializer(omp_priv = omp_orig)

#pragma omp parallel for num_threads(thread_count) reduction(vec_int_plus : y)
   for (int i=0; i<ROWS; i++) {
      for (int j=0; j<COLS; j++) {
         y[i] += A[i * COLS + j]*x[j];
      }
   }
}
```

Maticové operace

Násobení matice vektorem

- Zřejmá paralelizace – výpočet každé složky y je nezávislý a všechny mohou být spočteny paralelně

```
void multiply(std::vector<int>& A, std::vector<int>& x, std::vector<int>& y) {
#pragma omp declare reduction(vec_int_plus : std::vector<int> : \
                std::transform(omp_out.begin(), omp_out.end(), omp_in.begin(), omp_out.begin(), std::plus<int>())) \
                initializer(omp_priv = omp_orig)

#pragma omp parallel for num_threads(thread_count) reduction(vec_int_plus : y)
   for (int i=0; i<ROWS; i++) {
      for (int j=0; j<COLS; j++) {
         y[i] += A[i * COLS + j]*x[j];
      }
   }
}
```

- Co můžeme zlepšit?

Maticové operace

Násobení matice vektorem

- Zřejmá paralelizace – výpočet každé složky y je nezávislý a všechny mohou být spočteny paralelně

```
void multiply(std::vector<int>& A, std::vector<int>& x, std::vector<int>& y) {
#pragma omp declare reduction(vec_int_plus : std::vector<int> : \
                std::transform(omp_out.begin(), omp_out.end(), omp_in.begin(), omp_out.begin(), std::plus<int>())) \
                initializer(omp_priv = omp_orig)

#pragma omp parallel for num_threads(thread_count) reduction(vec_int_plus : y)
   for (int i=0; i<ROWS; i++) {
      for (int j=0; j<COLS; j++) {
         y[i] += A[i * COLS + j]*x[j];
      }
   }
}
```

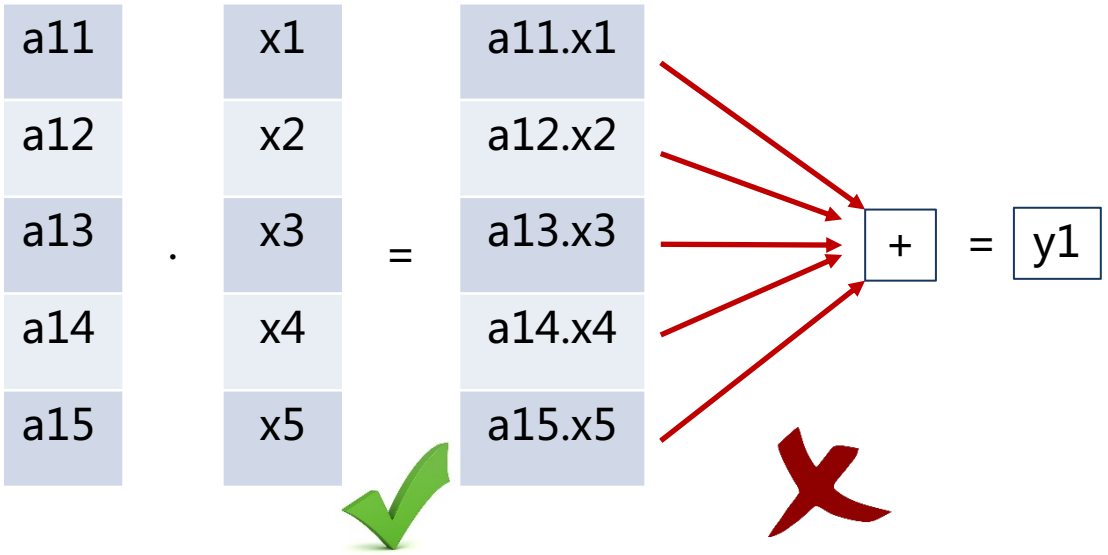
- Co můžeme zlepšit?
- Lze využít vektorizaci?

Maticové operace

Násobení matice vektorem

```
void multiply(std::vector<int>& A, std::vector<int>& x, std::vector<int>& y) {
#pragma omp declare reduction(vec_int_plus : std::vector<int> : \
                std::transform(omp_out.begin(), omp_out.end(), omp_in.begin(), omp_out.begin(), std::plus<int>())) \
                initializer(omp_priv = omp_orig)

#pragma omp parallel for num_threads(thread_count) reduction(vec_int_plus : y)
   for (int i=0; i<ROWS; i++) {
      for (int j=0; j<COLS; j++) {
         y[i] += A[i * COLS + j]*x[j];
      }
   }
}
```



Maticové operace

Násobení matice vektorem

- Co když budeme násobit po sloupcích?

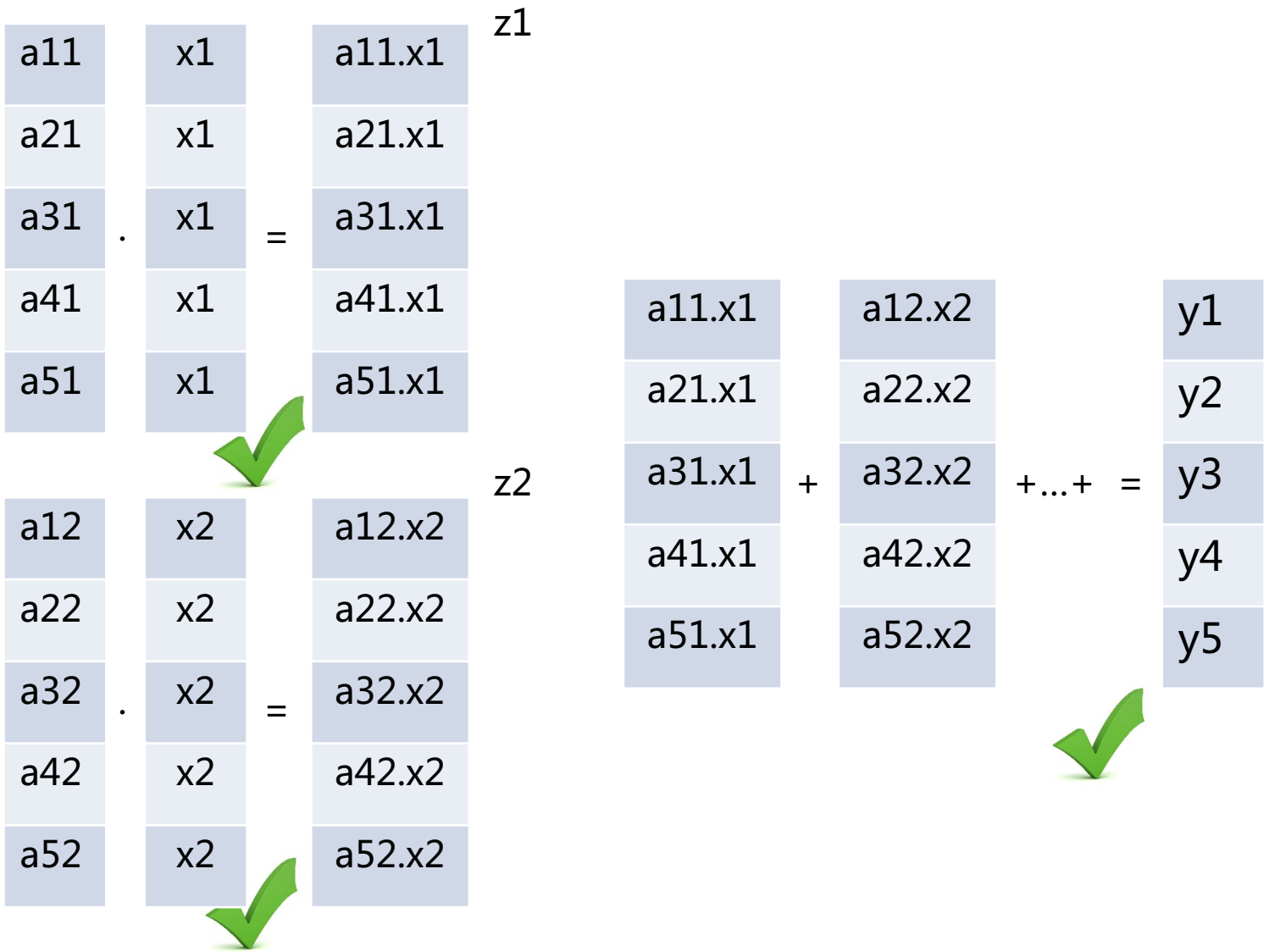
P1 P2 …

a11	a12	a13	a14	a15		x1		y1
a21	…					x2		y2
a31					X	x3	=	y3
a41						x4		y4
a51				a55		x5		y5

$$z_{ij} = a_{ij}.x_j \qquad y_i = \sum_{\{j=1,\ldots,5\}} z_{ij}$$

Maticové operace

Násobení matice vektorem



Maticové operace

Násobení matice vektorem

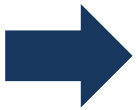
- Bude to fungovat?

Maticové operace

Násobení matice vektorem

- Bude to fungovat?
  - Operace sice mohou být vektorizovány, nicméně přístup není vhodný pro cache (mnoho dotazů)
  - Můžeme data v matici uspořádat po sloupcích

a11	a12	a13	a14	a15
a21	…			
a31				
a41				
a51				a55



a11	a21	a31	a41	a51	…		

Maticové operace

Násobení matice vektorem

- Bude to teď fungovat?

```
...
// data has to be ordered by columns in memory
for (int i = 0; i < COLS; i++) {
        x[i] = rand() % 1000;
        for (int j = 0; j < ROWS; j++) {
            A[i * ROWS + j] = rand() % 1000;
        }
    }
...

void multiply_column(std::vector<int> &A, std::vector<int> &x, std::vector<int> &y) {
#pragma omp declare reduction(vec_int_plus : std::vector<int> : \
            std::transform(omp_out.begin(), omp_out.end(), omp_in.begin(), omp_out.begin(), std::plus<int>())) \
            initializer(omp_priv = omp_orig)

#pragma omp parallel for num_threads(thread_count) reduction(vec_int_plus : y)
   for ( int i = 0; i < COLS ; i ++) {
      for (int j = 0; j < ROWS; j++) {
         y[j] += A[i * ROWS + j]*x[i];
      }
   }
}
```

Maticové operace

Násobení matice vektorem

- Lze dále zefektivnit původní přístup?
  - Vzpomeňte si na falsesharing …

```
void multiply(std::vector<int>& A, std::vector<int>& x, std::vector<int>& y) {
#pragma omp declare reduction(vec_int_plus : std::vector<int> : \
                std::transform(omp_out.begin(), omp_out.end(), omp_in.begin(), omp_out.begin(), std::plus<int>())) \
                initializer(omp_priv = omp_orig)

#pragma omp parallel for num_threads(thread_count) reduction(vec_int_plus : y)
   for (int i=0; i<ROWS; i++) {
      for (int j=0; j<COLS; j++) {
         y[i] += A[i * COLS + j]*x[j];
      }
   }
}
```

Maticové operace

Násobení matice vektorem

- Lze dále zefektivnit původní přístup?
  - Vzpomeňte si na falsesharing …

```
void multiply(std::vector<int>& A, std::vector<int>& x, std::vector<int>& y) {
#pragma omp declare reduction(vec_int_plus : std::vector<int> : \
                std::transform(omp_out.begin(), omp_out.end(), omp_in.begin(), omp_out.begin(), std::plus<int>())) \
                initializer(omp_priv = omp_orig)

#pragma omp parallel for num_threads(thread_count) reduction(vec_int_plus : y)
   for (int i=0; i<ROWS; i++) {
      for (int j=0; j<COLS; j++) {
         y[i] += A[i * COLS + j]*x[j];
      }
   }
}
```

- Nahradíme pole lokální proměnnou

Maticové operace

Násobení matice vektorem

- Lokální proměnná

```
void multiply(std::vector<int> &A, std::vector<int> &x, std::vector<int> &y) {
#pragma omp declare reduction(vec_int_plus : std::vector<int> : \
                std::transform(omp_out.begin(), omp_out.end(), omp_in.begin(), omp_out.begin(), std::plus<int>())) \
                initializer(omp_priv = omp_orig)

   int tmp;
#pragma omp parallel for num_threads(thread_count) reduction(vec_int_plus : y)
   for (int i=0; i<ROWS; i++) {
      tmp = 0;
      for (int j=0; j<COLS; j++) {
         tmp += A[i * COLS + j]*x[j];
      }
      y[i] += tmp;
   }
}
```

Maticové operace

Násobení matice vektorem

Computer Languages, Systems & Structures 51 (2018) 158–175

Contents lists available at ScienceDirect

Computer Languages, Systems & Structures

journal homepage: www.elsevier.com/locate/cl

Effective Implementation of Matrix–Vector Multiplication on Intel's AVX multicore Processor

Somaia A. Hassan[a,*], Mountasser M.M. Mahmoud[a], A.M. Hemeida[b], Mahmoud A. Saber[a]

Maticové operace

Násobení dvou matic

a11	a12	a13		b11	b12	b13		c11	c12	c13
a21	…		x	b21	…		=	c21	…	
…				…				…		

$$c_{ij} = \sum_{\{k=1,\ldots,n\}} a_{ik} \,.\, b_{kj}$$

Maticové operace

Násobení dvou matic

a11	a12	a13		b11	b12	b13		c11	c12	c13
a21	…		X	b21	…		=	c21	…	
…				…				…		

Výpočet prvků c je opět nezávislý a lze paralelizovat

$$c_{ij} = \sum_{\{k=1,\ldots,n\}} a_{ik} \,.\, b_{kj}$$

Nevýhody?
Velké množství úloh, malé úlohy

Maticové operace

Násobení dvou matic

a11	a12	a13
a21	…	
…		

X

b11	b12	b13
b21	…	
…		

=

c11	c12	c13
c21	…	
…		

Můžeme zvětšit úkoly spojením několika řádků

```
void multiply(std::vector<int>& A, std::vector<int>& B, std::vector<int>& C) {
#pragma omp declare reduction(vec_int_plus : std::vector<int> : \
                std::transform(omp_out.begin(), omp_out.end(), omp_in.begin(), omp_out.begin(), std::plus<int>())) \
                initializer(omp_priv = omp_orig)
   int tmp;
#pragma omp parallel for collapse(2) num_threads(thread_count) reduction(vec_int_plus : C)
   for (int i=0; i<ROWS; i++) {
      for (int j=0; j<COLS; j++) {
         tmp = 0;
         for (int k=0; k<ROWS; k++) {
            tmp += A[i * COLS + k] * B[k * COLS + j];
         }
         C[i * COLS + j] += tmp;
      }
   }
}
```

Maticové operace

Násobení dvou matic

- Rozdělení na bloky
- 1 úkol odpovídá částečnému výsledku submatice

Maticové operace

Násobení dvou matic

- Rozdělení na bloky
- 1 úkol odpovídá částečnému výsledku submatice (např. c11,c12,c21,c22)

b11	b12	b13	b14
b21	…		
…			

a11	a12	a13	a14
a21	…		
…			

c11	c12	c13	c14
c21	…		
…			

$$c_{11} \mathrel{+}= a_{11}.b_{11} + a_{12}.b_{21}$$
$$c_{12} \mathrel{+}= a_{11}.b_{12} + a_{12}.b_{22}$$
$$\dots$$

Maticové operace

Násobení dvou matic

```
void multiply_blocks(std::vector<int>& A, std::vector<int>& B, std::vector<int>& C) {
#pragma omp declare reduction(vec_int_plus : std::vector<int> : \
                std::transform(omp_out.begin(), omp_out.end(), omp_in.begin(), omp_out.begin(), std::plus<int>())) \
                initializer(omp_priv = omp_orig)


   const int ROWS_IN_BLOCK = 10;
   const int BLOCKS_IN_ROW = ROWS/ROWS_IN_BLOCK;
   int tmp;

#pragma omp parallel for collapse(2) num_threads(thread_count) reduction(vec_int_plus : C) private(tmp)
   for (int br1=0; br1<BLOCKS_IN_ROW; br1++) {
      for (int bb=0; bb<BLOCKS_IN_ROW; bb++) {
         for (int bc2 = 0; bc2 < BLOCKS_IN_ROW; bc2++) {
            for (int r = br1 * ROWS_IN_BLOCK; r < (br1 + 1) * ROWS_IN_BLOCK; r++) {
               for (int c = bc2 * ROWS_IN_BLOCK; c < (bc2 + 1) * ROWS_IN_BLOCK; c++) {
                  tmp = 0;
                  for (int k = 0; k < ROWS_IN_BLOCK; k++) {
                     tmp += A[r * COLS + (k + bb*ROWS_IN_BLOCK)] * B[(bb*ROWS_IN_BLOCK + k) * COLS + c];
                  }
                  C[r * COLS + c] += tmp;
               }
            }
         }
      }
   }
}
```

Maticové operace

Násobení dvou matic

a11	a12	a13
a21	…	
…		

X

b11	b12	b13
b21	…	
…		

=

c11	c12	c13
c21	…	
…		

- A co dál? Lze využít vektorizaci?
- Můžeme najednou spočítat vektor částečných hodnot?

Maticové operace

Násobení dvou matic

a11	a12	a13
a21	…	
…		

X

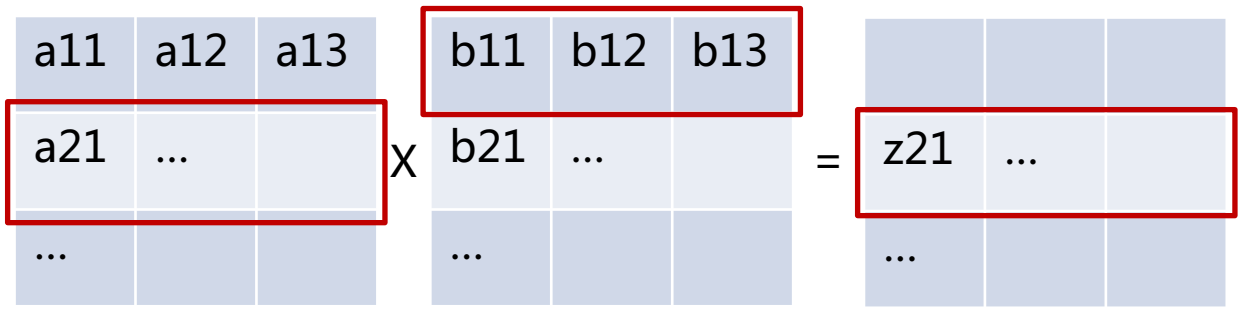
b11	b12	b13
b21	…	
…		

=

c11	c12	c13
c21	…	
…		

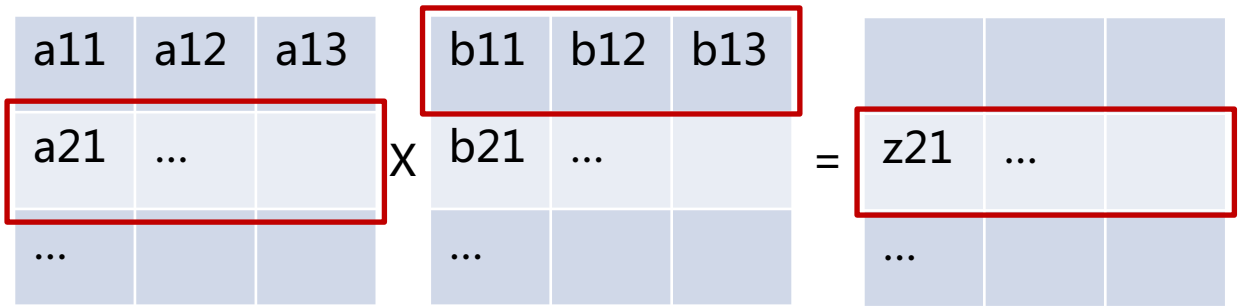
- A co dál? Lze využít vektorizaci?
- Můžeme najednou spočítat vektor částečných hodnot?

- Co když budeme násobit řádek *i* matice A a řádek *j* matice B?



Maticové operace

Násobení dvou matic



- Opět máme vektor dílčích výsledků *z*
- Násobení je vektorové, součet různých vektorů z lze také vektorizovat

```
void multiply(std::vector<int>& A, std::vector<int>& B, std::vector<int>& C) {
#pragma omp declare reduction(vec_int_plus : std::vector<int> : \
                std::transform(omp_out.begin(), omp_out.end(), omp_in.begin(), omp_out.begin(), std::plus<int>())) \
                initializer(omp_priv = omp_orig)

#pragma omp parallel for collapse(2) num_threads(thread_count) reduction(vec_int_plus : C)
   for (int r1=0; r1<ROWS; r1++) {
      for (int r2=0; r2<ROWS; r2++) {
         for (int k=0; k<ROWS; k++) {
            C[r1 * COLS + k] += A[r1 * COLS + r2] * B[r2 * COLS + k];
         }
      }
   }
}
```

Maticové operace

Gaussova eliminace

- Dalším typickým úkolem je řešení soustavy lineárních rovnic
- Lze využít Gaussovu eliminaci

a11	a12	a13	b1
a21	…		b2
…			b3

a11	a12	a13	b1
0	a´22	a´23	b´2
0	0	a´33	b´3

- Jak můžeme paralelizovat?

Maticové operace

Gaussova eliminace

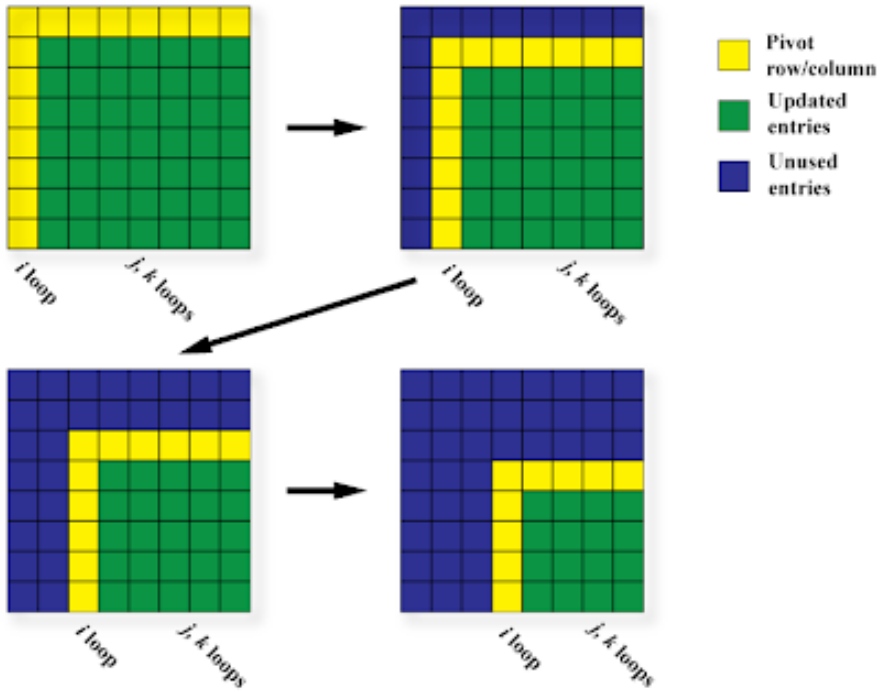
- Jaké jsou závislosti mezi hodnotami?
  - Po výběru pivota se změní všechny hodnoty pro řádky a sloupce větší než pozice pivota

Maticové operace

Gaussova eliminace

- Jaké jsou závislosti mezi hodnotami?
  - Po výběru pivota se změní všechny hodnoty pro řádky a sloupce větší než pozice pivota
- Kterou část můžeme paralelizovat?

```
void gauss(std::vector<double>& A) {
    for (int i=0; i<ROWS; i++) {
        // Make all rows below this one 0 in current column
        for (int k=i+1; k<ROWS; k++) {
            double c = -A[k * COLS + i]/A[i*COLS + i];
            for (int j=i; j<ROWS; j++) {
                if (i==j) {
                    A[k * COLS + j] = 0;
                } else {
                    A[k * COLS + j] += c * A[i * COLS + j];
                }
            }
        }
    }
}
```



Maticové operace

Gaussova eliminace

- Jaké jsou závislosti mezi hodnotami?
  - Po výběru pivota se změní všechny hodnoty pro řádky a sloupce větší než pozice pivota

```
void gauss_par(std::vector<double>& A) {
#pragma omp declare reduction(vec_int_plus : std::vector<double> : \
                std::transform(omp_out.begin(), omp_out.end(), omp_in.begin(), omp_out.begin(), std::plus<double>())) \
                initializer(omp_priv = omp_orig)

   for (int i=0; i<ROWS; i++) {
      // Make all rows below this one 0 in current column
#pragma omp parallel for num_threads(thread_count)
      for (int k=i+1; k<ROWS; k++) {
         double c = -A[k * COLS + i]/A[i*COLS + i];
         for (int j=i; j<ROWS; j++) {
            if (i==j) {
               A[k * COLS + j] = 0;
            } else {
               A[k * COLS + j] += c * A[i * COLS + j];
            }
         }
      }
   }
}
```

Maticové operace

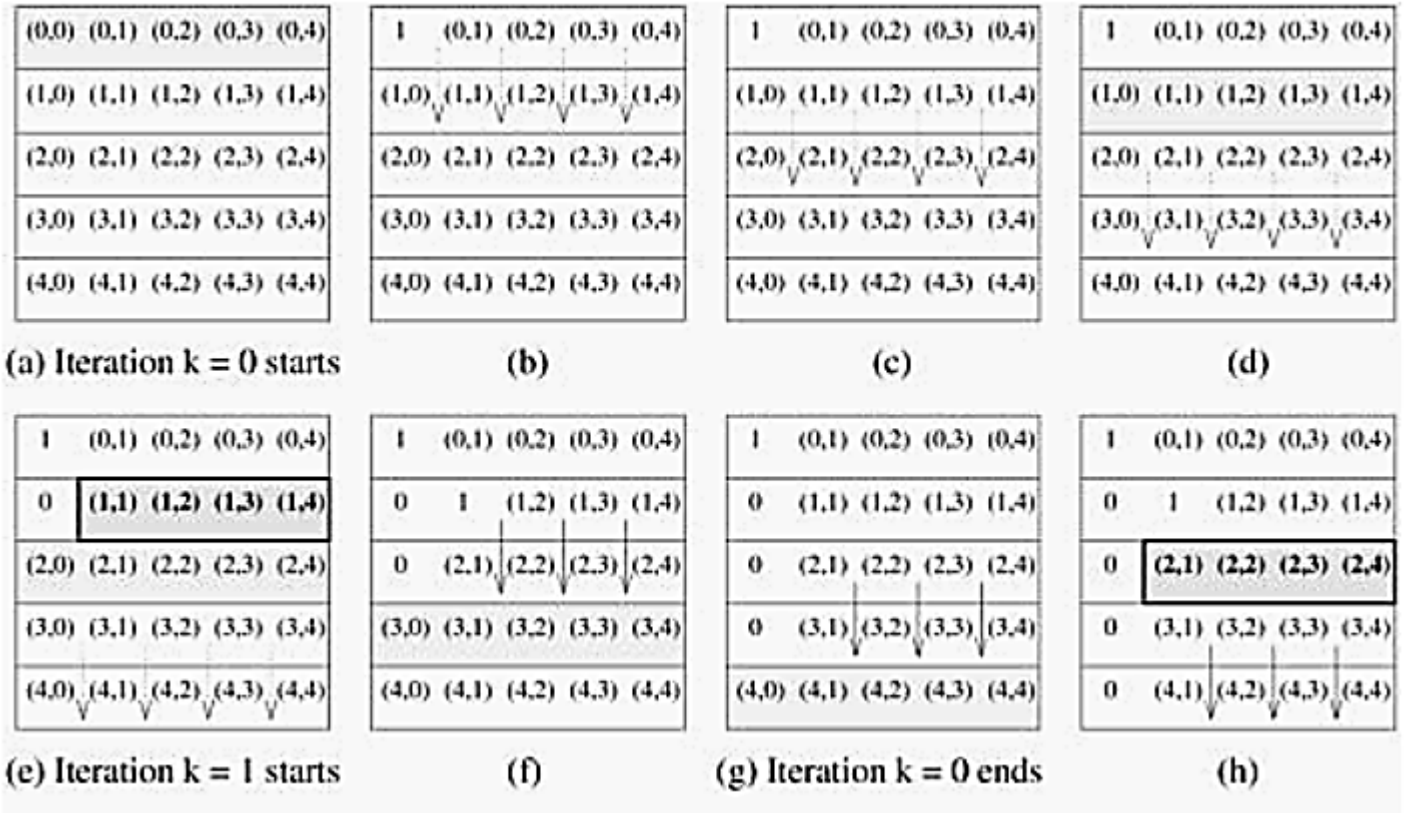
Gaussova eliminace

- Co lze dále zefektivnit?

Maticové operace

Gaussova eliminace

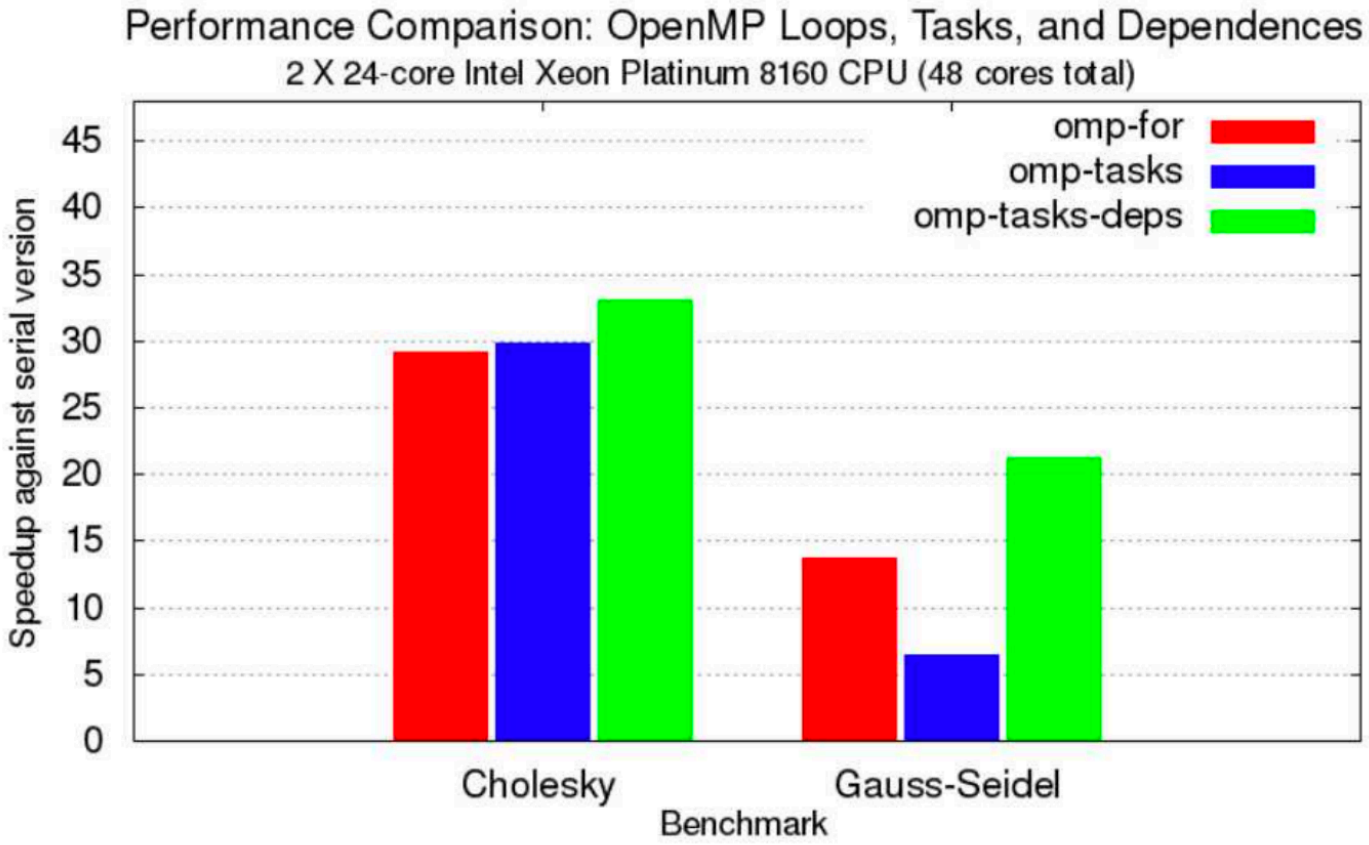
- Co lze dále zefektivnit?



OpenMP: Novinky za posledních 5 let

Ze třetí přednášky

- OpenMP je specifikace, která se vyvíjí (podobně jako C++):



"The Ongoing Evolution of OpenMP"
https://www.osti.gov/pages/servlets/purl/1465188

OpenMP: Novinky za posledních 5 let

Offloading na GPGPU:

```
#pragma omp target data map (to: pA[0:N*N],pB[0:N*N]) map
(tofrom: pC[0:N*N])
#pragma omp target
#pragma omp teams distribute parallel for collapse(2)
private(i,j,k)
for (i=0;i<N;i++) {
    for (j=0;j<N;j++) {
        for (k=0;k<N;k++) {
            pC(i,j)+=pA(i,k)*pB(k,j);
        }
    }
}
```

https://developer.ibm.com/technologies/systems/articles/gpu-programming-with-openmp/

Shrnutí paralelní části

- Základní nástroje pro psaní paralelního programu
  - vlákna a práce s nimi
  - Atomické proměnné
  - OpenMP
  - (Vektorizace, SIMD paralelizace)

Shrnutí paralelní části

- Základní nástroje pro psaní paralelního programu
  - vlákna a práce s nimi
  - atomické proměnné
  - OpenMP
  - Vektorizace (SIMD paralelizace)

- Základní techniky paralelizace
  - Rozděluj a panuj
  - Threadpool
  - Dekompozice, nalezení co možná nejvíc paralelně bezkonfliktně vykonatelných operací

Shrnutí paralelní části

- Základní nástroje pro psaní paralelního programu
  - vlákna a práce s nimi
  - Atomické proměnné
  - OpenMP
  - (Vektorizace, SIMD paralelizace)

- Základní techniky paralelizace
  - Rozděluj a panuj
  - Threadpool
  - Dekompozice, nalezení co možná nejvíc paralelně bezkonfliktně vykonatelných operací

- Základní algoritmy
  - Řazení
  - Maticové operace

Shrnutí paralelní části

- Paralelizujete s cílem zefektivnit běh programu/algoritmu
- Musíte (alespoň částečně) rozumět vykonávání programu na HW
  - Falsesharing
  - Cache optimization

Shrnutí paralelní části

- Paralelizujete s cílem zefektivnit běh programu/algoritmu
- Musíte (alespoň částečně) rozumět vykonávání programu na HW
  - Falsesharing
  - Cache optimization

- Vynechali jsme spoustu věcí
  - Úvodní kurz, aby jste získali základní znalosti a zkušenosti

- Pokud Vás paralelní programování zaujalo
  - Paralelní algoritmy (B4M35PAG)
  - Obecné výpočty na grafických procesorech (B4M39GPU)

Shrnutí paralelní části

- Pro implementační zkoušku – programujte, programujte, programujte!

Shrnutí paralelní části

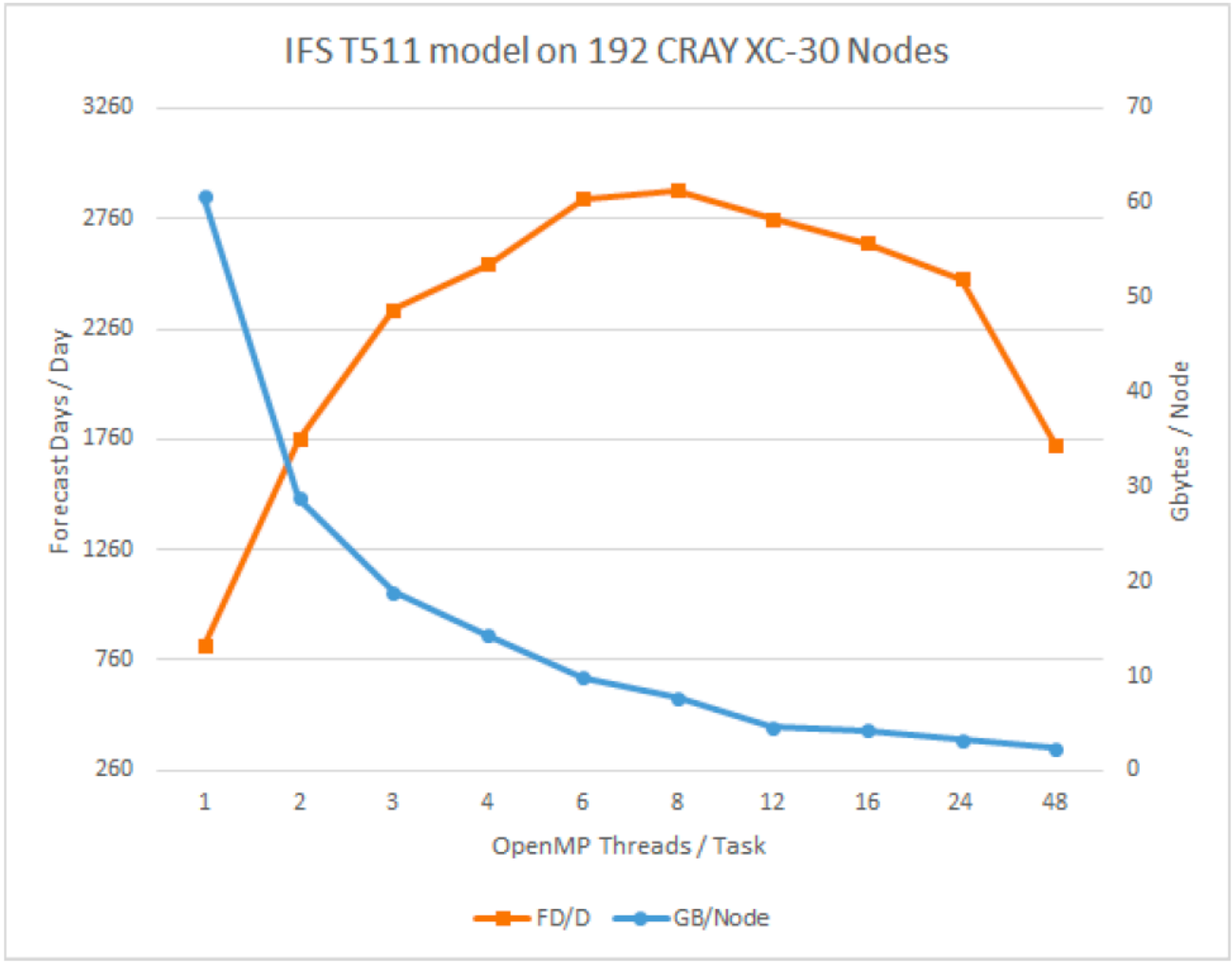
- Pro implementační zkoušku – programujte, programujte, programujte!
  - Dostanete problém + sériový algoritmus
  - Cílem bude **zrychlit** algoritmus **díky paralelizaci**
    - **Paralelizace musí být korektní!** (musíte přemýšlet, ne všechny chyby se projeví, paralelní programování je nedeterministické)

Shrnutí paralelní části

- Pro implementační zkoušku – programujte, programujte, programujte!
  - Dostanete problém + sériový algoritmus
  - Cílem bude **zrychlit** algoritmus **díky paralelizaci**
    - **Paralelizace musí být korektní!** (musíte přemýšlet, ne všechny chyby se projeví, paralelní programování je nedeterministické)

- V dalším studiu/práci – paralelizujte, pokud je to potřeba!
  - Pracujte iterativně – nejdřív je potřeba mít korektní sériovou variantu
  - Pokud je pomalá – zrychlujeme, paralelizujeme, atd.

Shrnutí paralelní části



https://confluence.ecmwf.int/download/attachments/46600240/Intro_Parallel_Computing.pdf?api=v2