High Performance Parallel Beam and Perspective Cone-Beam Backprojection on Intel Xeon Phi

Matthias Baer[1,2] and Marc Kachelrieß[1,2]

[1]German Cancer Research Center (DKFZ), Heidelberg, Germany

[2]Friedrich-Alexander-University (FAU), Erlangen-Nürnberg, Germany

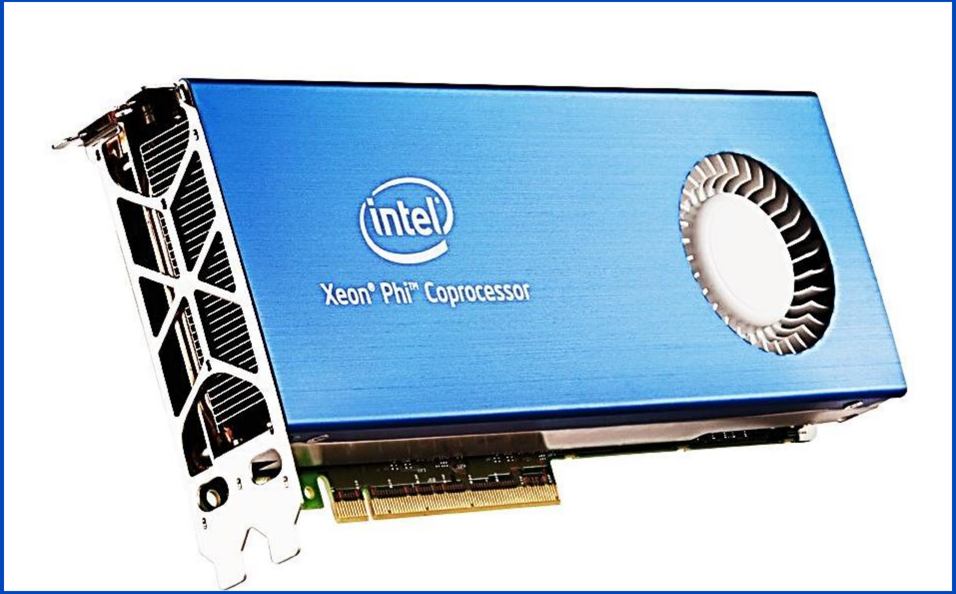
DEUTSCHES KREBSFORSCHUNGSZENTRUM IN DER HELMHOLTZ-GEMEINSCHAFT

Aim

To introduce some of the basic hardware and software features of Intel‘s new many core coprocessor Xeon Phi and implement an optimize parallel beam and perspective cone-beam backprojection on Intel‘s Xeon Phi.

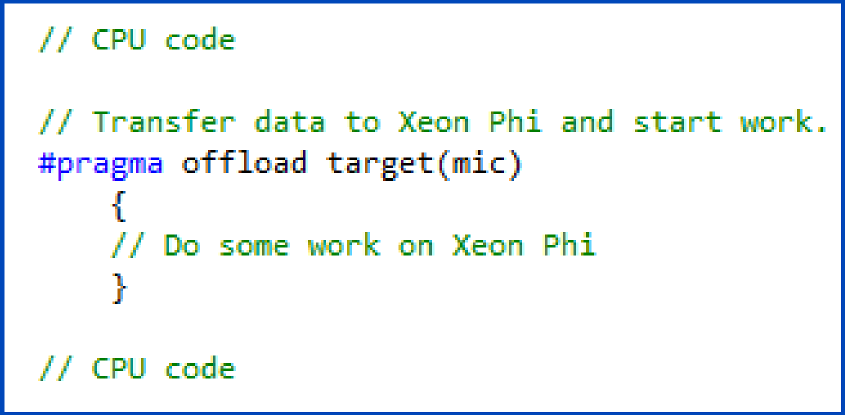
Hardware Properties

- Coprocessor connected via the PCIe bus to a host PC
- 61 Pentium-like x86 cores (pre–production engineering hardware sample, stepping B0)
  - Core frequency 1.2 GHz
  - 4 threads per core i.e., 244 threads in total
- 8 GB RAM
- One vector unit per core with 32 512 bit vector registers
- Peak performance 2 TFLOPS
- Price 2700 $
- Several Xeon Phis can run in a single host PC



Programming Model

- Programming for Xeon Phi can be done using either C/C++ or Fortran, no special programming language needed.
- Regions in the program that should be executed on Xeon Phi are marked by a simple pragma directive:

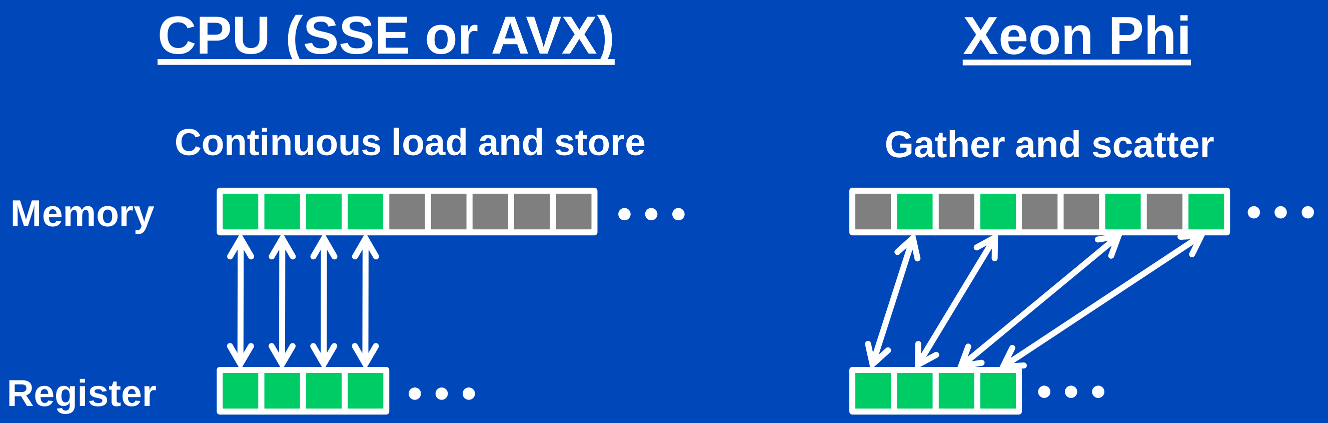


```
// CPU code

// Transfer data to Xeon Phi and start work.
#pragma offload target(mic)
    {
    // Do some work on Xeon Phi
    }

// CPU code
```

- Parallelism can be added for example by using OpenMP.
- A key factor to achieve a high performance on Xeon Phi is vectorization:
  - Xeon Phi has 512 bit vector registers, modern CPUs typically have 128 bit or 256 bit registers.
  - On Xeon Phi 16 floats or 8 doubles can be processed at once, two respectively four times more than on modern CPUs.
  - Xeon Phi‘s vector instruction set is more flexible than the SSE (128 bit) or AVX (256 bit) instruction set on modern Intel CPUs.

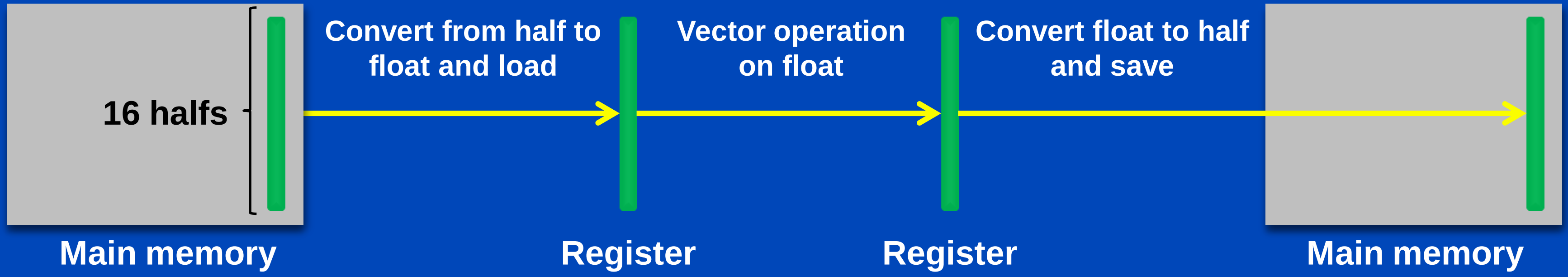
Vector Instruction Set

- Interesting new features of the Xeon Phi instruction set:
  - Gather and scatter: Xeon Phi supports to load / store data from / to arbitrary memory locations.
    Current CPUs support load and store from continuous memory locations only

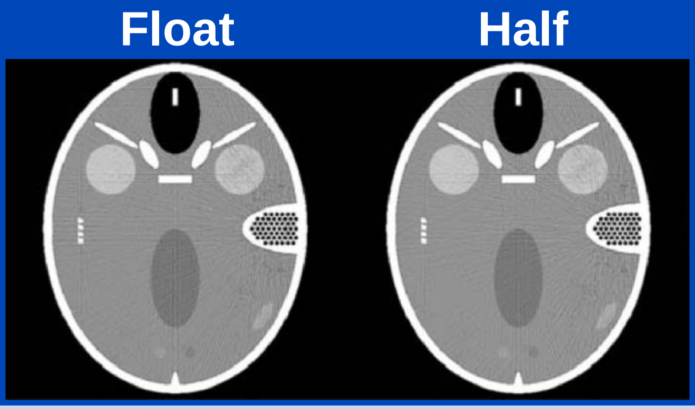


Vector Instruction Set

- Some interesting new features of the Xeon Phi instruction set are:
  - Xeon Phi supports the 16 bit floating point format (half).


16 halfs
Convert from half to float and load
Vector operation on float
Convert float to half and save
Main memory
Register
Register
Main memory

  - Using halfs instead of floats reduces the pressure on the memory bandwidth.
  - It was shown that using halfs during the reconstruction does not impair image quality [1].



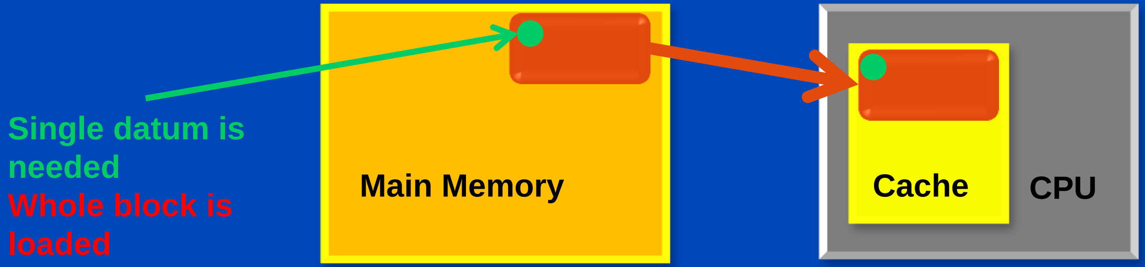
[1] C. Maaß, M. Baer, and M. Kachelrieß, CT image reconstruction with half precission floating-point values, Med.Phys. 38(2), 656-667.

Backprojection on Xeon Phi

- Parallel beam and perspective cone-beam backprojection
- Optimization techniques used in the implementations
  - Optimization of the data layout for the volume and the projection data
  - Parallelization over all available threads
  - Vectorization of the backprojection algorithms
  - Loop unrolling to hide instruction latency
  - Halfs instead of floats to represent the volume and projection data

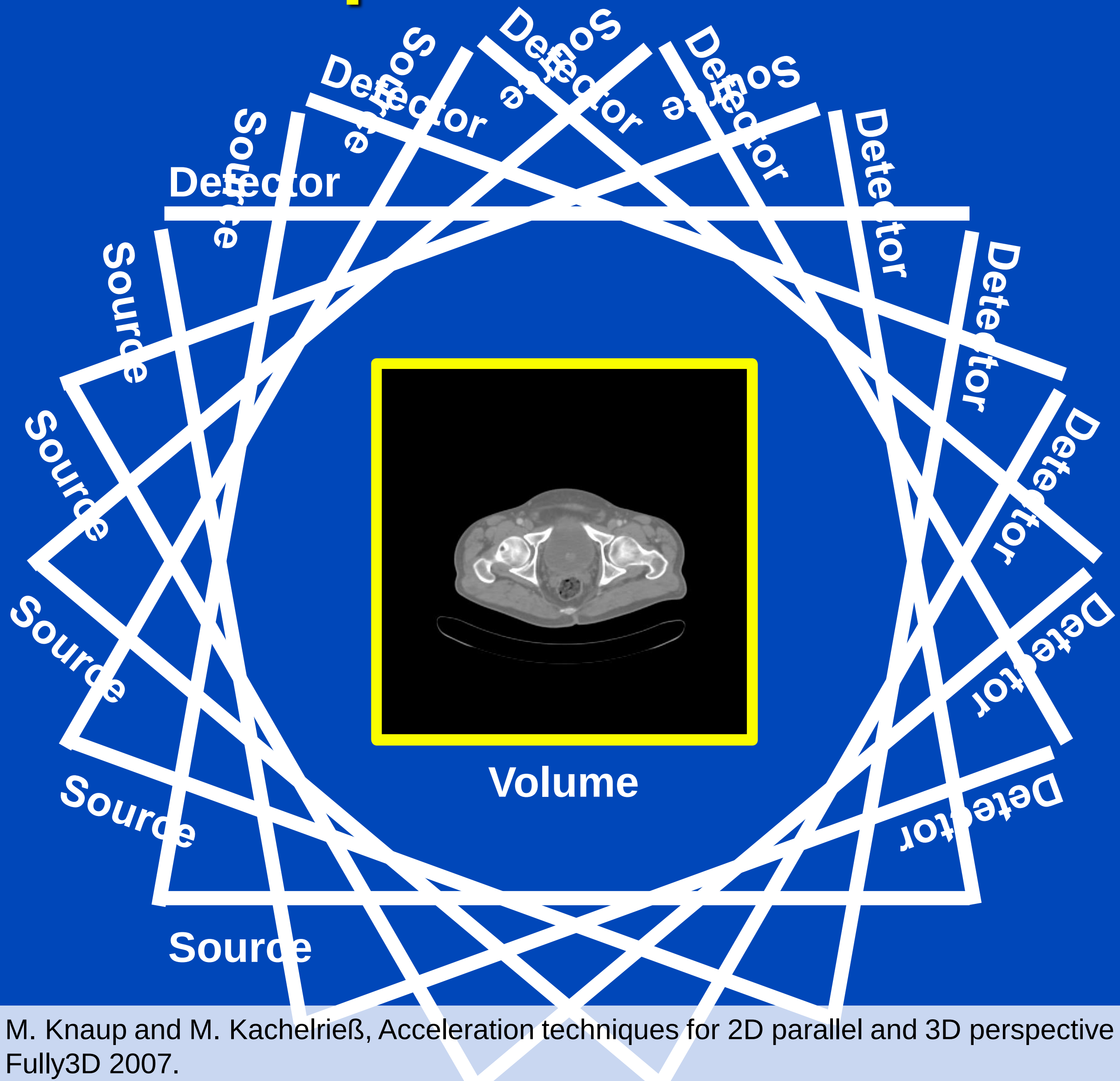
Optimization of Data Layout: Enforce Data Locality

- Typically backprojection has high demands on memory bandwidth
  - Low number of arithmetic operations
  - High number of memory operations
- Before data can be processed it needs to be loaded from main memory into the processors cache
  - Cache: Very fast memory close to the processor
  - Loading data from main memory into the cache takes a lot of time
    - » Memory access approx. 300 clock cycles, instruction latency 4 clock cycles
  - Data are loaded in blocks from main memory into cache



- To optimize cache usage try to localize the memory accesses

Optimization of Data Layout

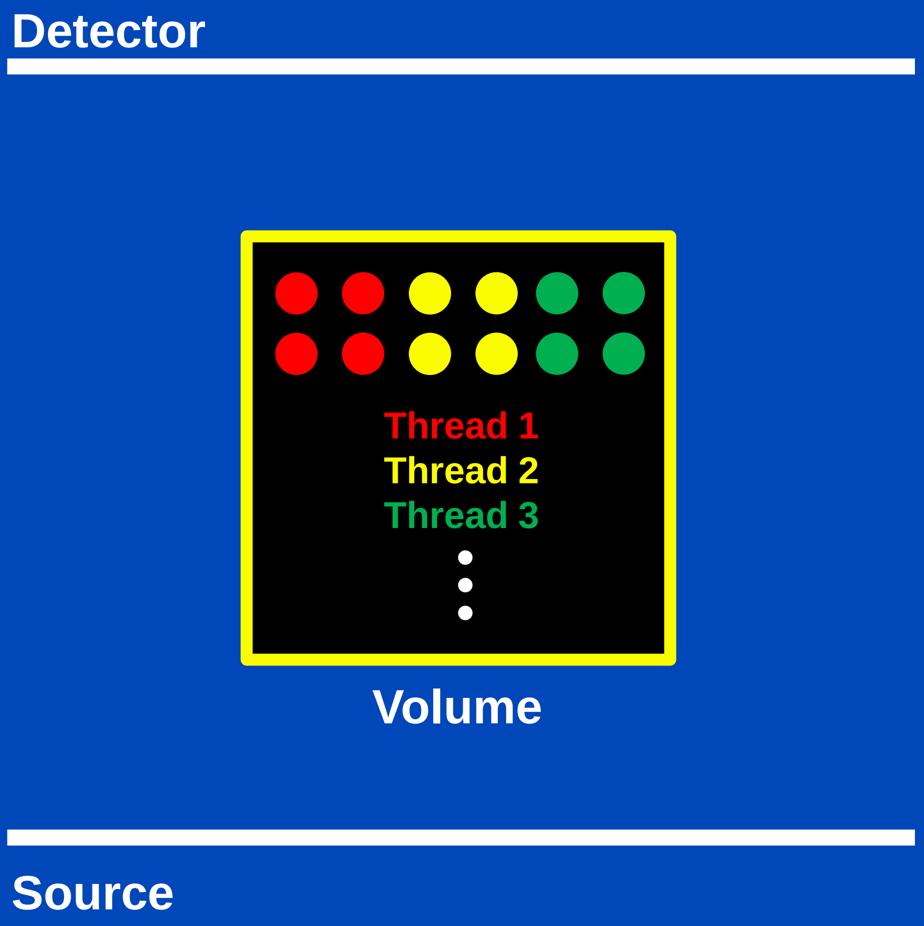


<u>Optimized approach</u>:

Reconstruct sub volumes from sub projections to avoid cache misses and to optimize the performance of the backprojection.

After the image has been updated from at least 180° the final image is obtained.

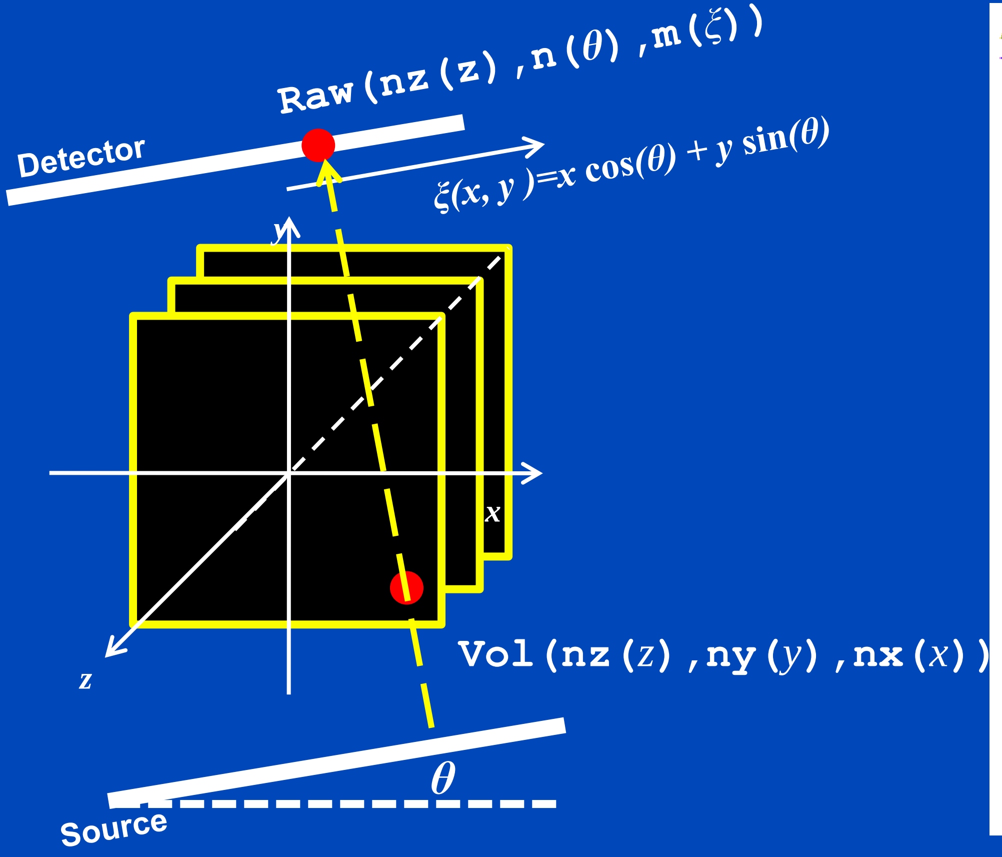
Parallelization

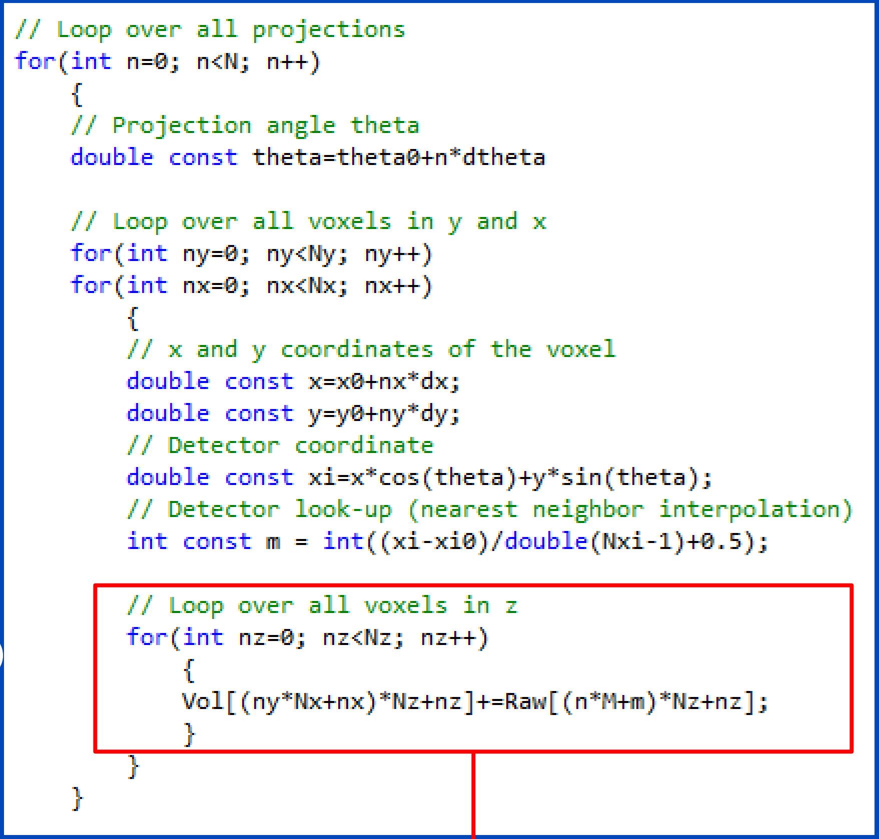


Parallelization is done over the sub volumes.

Each thread backprojects a single sub volume.

Vectorization – Parallel Beam Backprojection





```
// Loop over all projections
for(int n=0; n<N; n++)
    {
    // Projection angle theta
    double const theta=theta0+n*dtheta

    // Loop over all voxels in y and x
    for(int ny=0; ny<Ny; ny++)
    for(int nx=0; nx<Nx; nx++)
        {
        // x and y coordinates of the voxel
        double const x=x0+nx*dx;
        double const y=y0+ny*dy;
        // Detector coordinate
        double const xi=x*cos(theta)+y*sin(theta);
        // Detector look-up (nearest neighbor interpolation)
        int const m = int((xi-xi0)/double(Nxi-1)+0.5);

        // Loop over all voxels in z
        for(int nz=0; nz<Nz; nz++)
            {
            Vol[(ny*Nx+nx)*Nz+nz]+=Raw[(n*M+m)*Nz+nz];
            }
        }
    }
```

n:	**Projection index**
m:	**Detector pixel index**
nx, ny, nz:	**Voxel index in** x, y, z

Same operation for all slices nz, i.e. operation can be vectorized

Vectorization – Parallel Beam Backprojection

Reference code

```
// Loop over all voxels in z
for(int nz=0; nz<Nz; nz++)
    {
    Vol[(ny*Nx+nx)*Nz+nz]+=Raw[(n*M+m)*Nz+nz];
    }
```

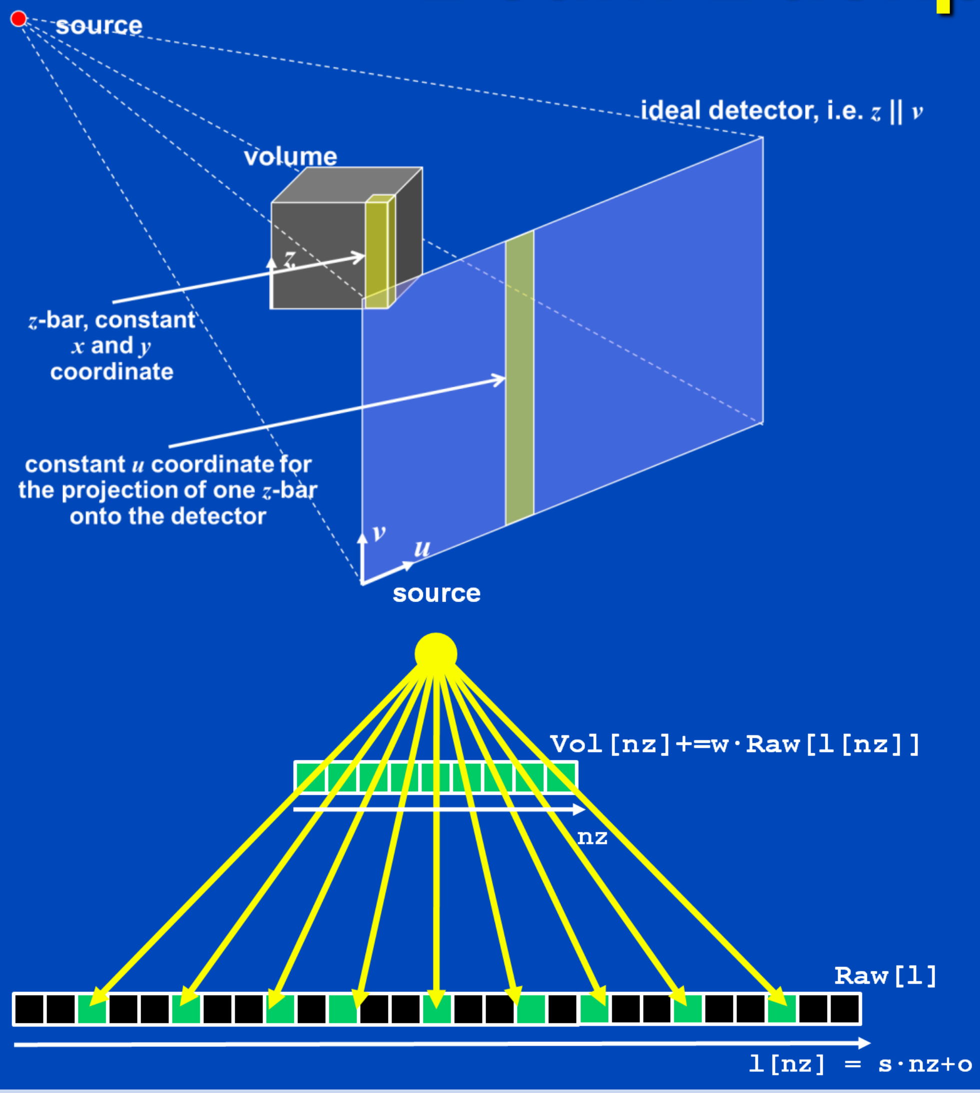
Vectorized Xeon Phi code

```
// Loop over all voxels in z
for(int nz=0; nz<Nz; nz+=16)
    {
    // Pointer to raw data and volume
    float * const RawPtr=&Raw[(n *M + m)*Nz+nz];
    float * const VolPtr=&Vol[(ny*Nx+nx)*Nz+nz];
    // Load projection data into register
    __m512 RawVec=_mm512_load_ps(RawPtr);
    // Load volume data into register
    __m512 VolVec=_mm512_load_ps(VolPtr);
    // Add projection data to volume
    VolVec=_mm512_add_ps(VolVec, RawVec);
    // Store result back to memory
    _mm512_store_ps(VolPtr ,VolVec);
    }
```

With Xeon Phi's vector unit 16 slices can be reconstructed simultaneously (would be 4 on a CPU using SSE and 8 using AVX).

1. **Load image data into register**
2. **Load raw data into register**
3. **Add raw data to image**
4. **Store image data back to memory**

Vectorization – Perspective Cone-Beam Backprojection



source
ideal detector, i.e. $z \parallel v$
volume
z
z-bar, constant x and y coordinate
constant u coordinate for the projection of one z-bar onto the detector
v
u
source
Vol[nz]+=w·Raw[l[nz]]
nz
Raw[l]
l[nz] = s·nz+o

The loop over nz, i.e. the loop over the z coordinate of the volume is a rescaling (scaling and resampling) operation.

The irregular data access pattern in this loop prohibits a vectorization on standard CPUs since here only loads and stores from continuous memory locations are supported

Xeon Phi's vector unit supports gather and scatter, i.e. loads and stores from arbitrary memory locations.

The nz loop can be vectorized on Xeon Phi, i.e. 16 xy-slices of the volume can be backprojected simultaneously.

Performance Measurements

- **As performance measure giga updates per second (GUPS) was used [1].**
- **The number of updates to backproject a volume is given by the number of accesses to the volume needed for backprojection (e.g. the backprojection of 512^3 voxels from 512 projections results in 512^4 updates (=64 Giga updates))**
- **Test cases**
  - **Parallel beam backprojection: 512 projections, 512^3 volume (64 giga updates)**
  - **Perspective cone-beam backprojection: 720 projections, 512^3 volume, (85 giga updates)**
  - **Linear interpolation on the detector**
  - **Float and half data format for the projection data and the volume**
- **The performance of the parallel beam and perspective cone-beam backprojection on Xeon Phi was compared with the performance on the GPU and the CPU.**
  - **GPU: NVIDIA Quadro 6000 GPU**
  - **CPU: PC equipped with two Intel Xeon E3-2670 processors, 2.6 GHz, 8 cores per processor, two threads per core, 256 bit vector registers**
- **The implementations for CPU [2] and GPU [3,4] were also highly optimized.**

[1] I. Goddart et al., Evolution of computer technology for fast cone beam backprojection, Computational Imaging Conference 2007.
[2] M. Knaup and M. Kachelrieß, Acceleration techniques for 2D parallel and 3D perspective forward- and backprojections, Fully3D 2007.
[3] M. Knaup and M. Kachelrieß, GPU-based parallel-beam and cone-beam forward- and backprojection using CUDA, IEEE Medical Imaging Conference Record 2008,.
[4] S. Sawall, L. Ritschl, M. Knaup, and M. Kachelrieß, Performance comparison of OpenCL and CUDA by benchmarking an optimized perspective backprojection, Fully3D 2011.

Performance Parallel Beam Backprojection

Xeon Phi (floats)	Xeon Phi (halfs)	CPU (floats)	GPU (floats)
58 GUPS	81 GUPS	51 GUPS	25 GUPS

- **Xeon Phi reaches about the same performance as the CPU when the volume and the projection data are represented by floats.**
- **Strong impact of half due to the bandwidth limitation of the parallel beam backprojection**
- **Xeon Phi and the CPU outperform the GPU (NVIDIA Quadro 6000).**

Performance Perspective Cone-Beam Backprojection

Xeon Phi (floats)	Xeon Phi (halfs)	CPU (floats)	GPU (floats)
27 GUPS	31 GUPS	7 GUPS	25 GUPS

- **Xeon Phi outperforms the CPU at least by a factor of three.**
- **Performance on Xeon Phi increases only slightly (≈10%) when switching from float to half.**
  - **Lower impact due to a lower ratio of memory operations to arithmetic operations as compared to the parallel beam backprojection**
  - **The usage of halfs has still the benefit of a lower overall memory consumption, e.g. larger volumes fit into the RAM of Xeon Phi.**
- **Xeon Phi reaches about the same performance as the GPU (NVIDIA Quadro 6000).**

Summary and Conclusion

- Xeon Phi is a highly parallel architecture (many cores, large vector registers).
- Porting existing C/C++ code to Xeon Phi can be achieved with only minor modifications and the more flexible vector instruction set (e.g. scatter and gather) may allow to vectorize algorithms that are unvectorizable on the CPU.
- Implemented and optimized a parallel beam and perspective cone-beam backprojection for Xeon Phi.
  - Memory layout, parallelization, vectorization, halfs for the volume and the projection data
- The CPU and the GPU have clear favorites in terms of backprojection algorithms.
  - CPU is fast for parallel beam backprojection
  - GPU is fast for perspective cone-beam backprojection
- Xeon Phi is competitive with both platforms even in their best cases.
- Xeon Phi may be an alternative for reducing computation times for complex algorithms in medical imaging.

Thank you.

This presentation will be soon available at www.dkfz.de/ct

This project was supported by a grant from the Intel Corporation.