UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

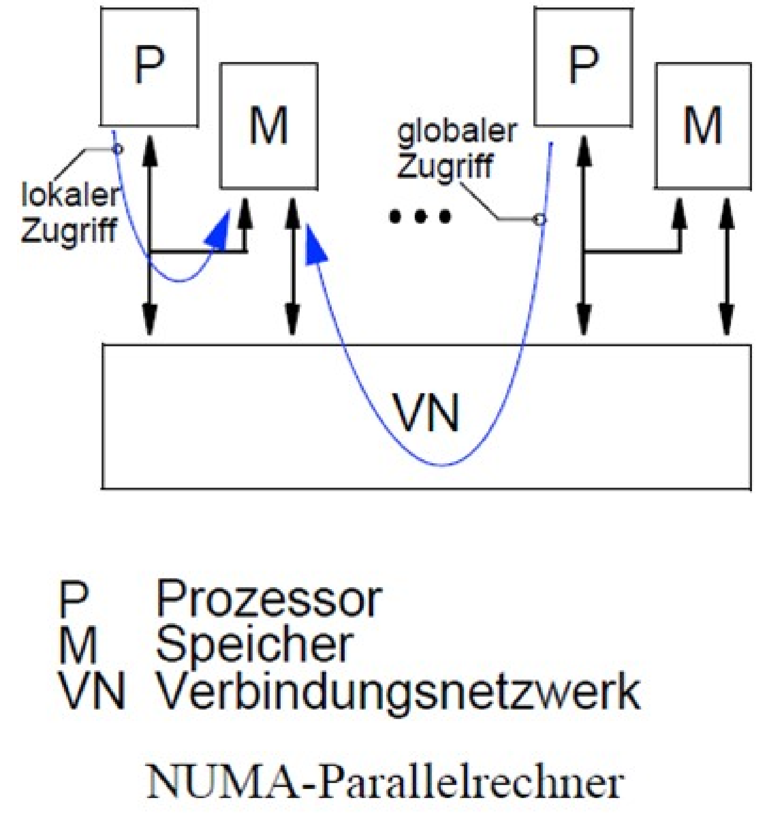
Project: NUMA Migration

Martin Wenzel,
Dennis Sebastian Rieber

Structure

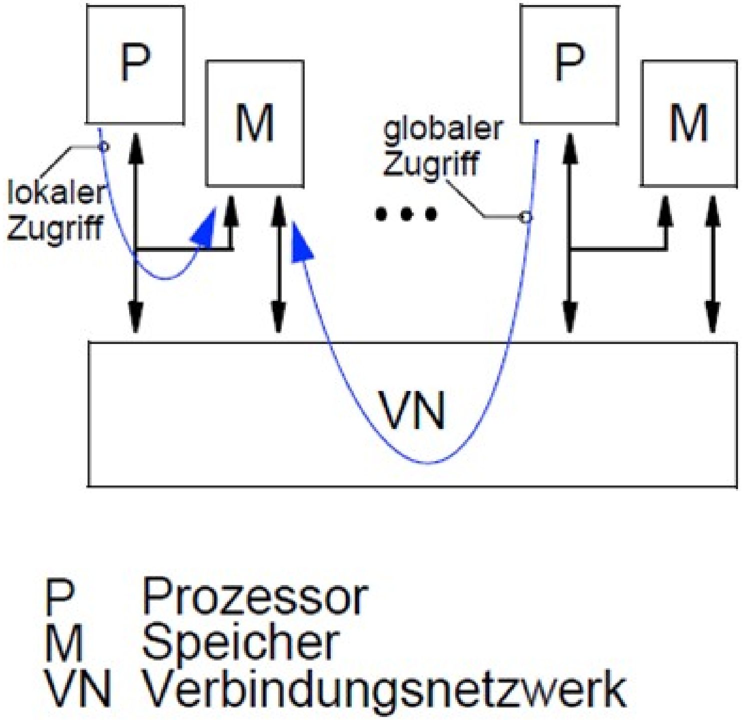
What is NUMA?

- Non-Uniform Memory Architecture
- Multisocket Systems
- shared memory systems
- cache coherent



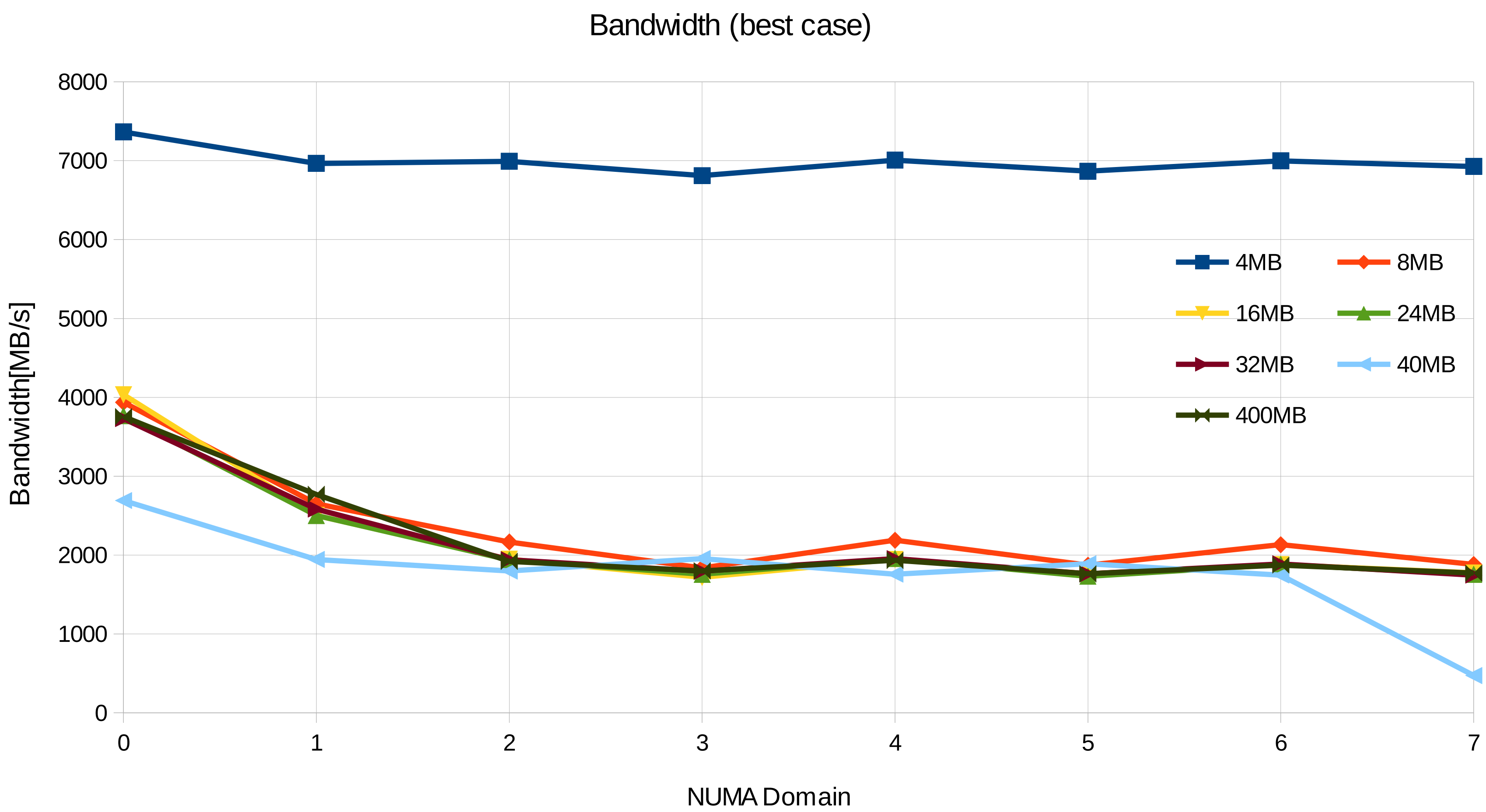
Motivation – Why are doing this?

- Remote data access
- first touch policy
  - Employed by all modern operating systems
  - Initializes memory page on the domain of the thread, which is the first to touch the data



NUMA-Parallelrechner

Motivation – Costs of remote memory access



State of the Art – What else is out there?

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

- Hwloc – hardware locality
  - Explores system architecture
  - Creates architectural model of the system
  - Huge interface
- Libnuma
  - Only available on Unix systems
  - Classic C style interface
  - Memory page and thread migration possibilities

State of the Art – libnuma

```
#include <numa.h>
int         _pagesize           = sysconf(_SC_PAGESIZE);
int         _numa_pagecount  = _size * sizeof(float) /_pagesize;
int*        _numa_destination       = new int[_numa_pagecount];
int*        _status          = new int[_numa_pagecount];
void**     _numa_pages  = (void**) malloc (_numa_pagecount
*sizeof(void *));

for (int i = 0; i < _numa_pagecount; i++){
      _numa_pages[i] = &_data[i*(_pagesize / sizeof(float))];
      _numa_destination[i] = domain;
}

if (move_pages(0, _numa_pagecount, _numa_pages,
_numa_destination,  _status, MPOL_MF_MOVE) != 0) {
      std::cout << "something went wrong"<< std::endl;
}
```

Structure

- Introduction
  - What is NUMA?
  - Motivation – Why are we doing this?
  - State of the Art – What else is out there?
- Realization – What can we do about the Problem?
  - Goals and Concepts
  - Interface
  - Implementation
- Verification – Was it all worth it?
  - Migration
  - Stencil
- Summary

Goals

UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

- Core
  - ✓ Container
  - ✓ First touch migration of complete data block
- Target
  - ✓ Distribute pages over several domains
  - ✓ Templated Container
- Enhanced
  - ✗ Intelligent distribution based on access patterns
  - ✗ Complete portability

Concepts

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

- NUMA aware container class
  - Knowing the available domains
  - Move data between domains
- STL-like behavior
  - Iterators
  - Support foreach Syntax
  - Templates
- Using first-touch to move data, no syscalls required
- easy user Interface with c++ language features

Realization - Interface

```
template <class T> class numavec

    typedef typename std::vector<T>::iterator iterator;
    typedef typename std::vector<T>::const_iterator const_iterator
    typedef std::initializer_list<int> init;

    numavec<T>( int );
     numavec<T>(int, std::map<int, int> &);
    ~numavec<T>() = default;

    numavec<T>( const numavec& ); // copy
    numavec<T>( numavec&& ); // move
    numavec<T>& operator=( const numavec& ); // copy
    numavec<T>& operator=( numavec&& ); // move

    T& operator[](int in) { return _data[in]; };
    T& at(int);

    int size() const { return _data.size(); };
    const T* data() const { return _data; };

    void migrate_to(int);
    void distribute( init );
    void uneven_distribute( std::map<int,int> &);

    const_iterator cbegin() const { return _data.cbegin(); };
    const_iterator cend() const { return _data.cend() + _size; };

    const_iterator begin() const {return _data.begin();};
    const_iterator end() const { return _data.end() + _size; };

    iterator begin() {return _data.begin();};
    iterator end() { return _data.end() + _size;};
```

Realization

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

```
_core_pin( _domain_core[_destination.begin()->second]);

std::vector<T> new_data;

new_data.swap(_data);

_data.clear();
_data.shrink_to_fit();
_data.reserve(_size);

std::vector<std::thread> threads(_destination.size());
    int pos1 = 0; int i = 0;
    for (auto& e: _destination){ //-- iterate each interval
   threads[i] = std::thread(&numavec::_migrate_part,this, e.second, pos1,
           e.first, std::ref(new_data));
           pos1 = e.first +1; //-- define the next start
   ++i;
    }
for ( unsigned int i = 0; i < threads.size(); i++ ) {
   threads[i].join();
}
```

Realization – Thread pin

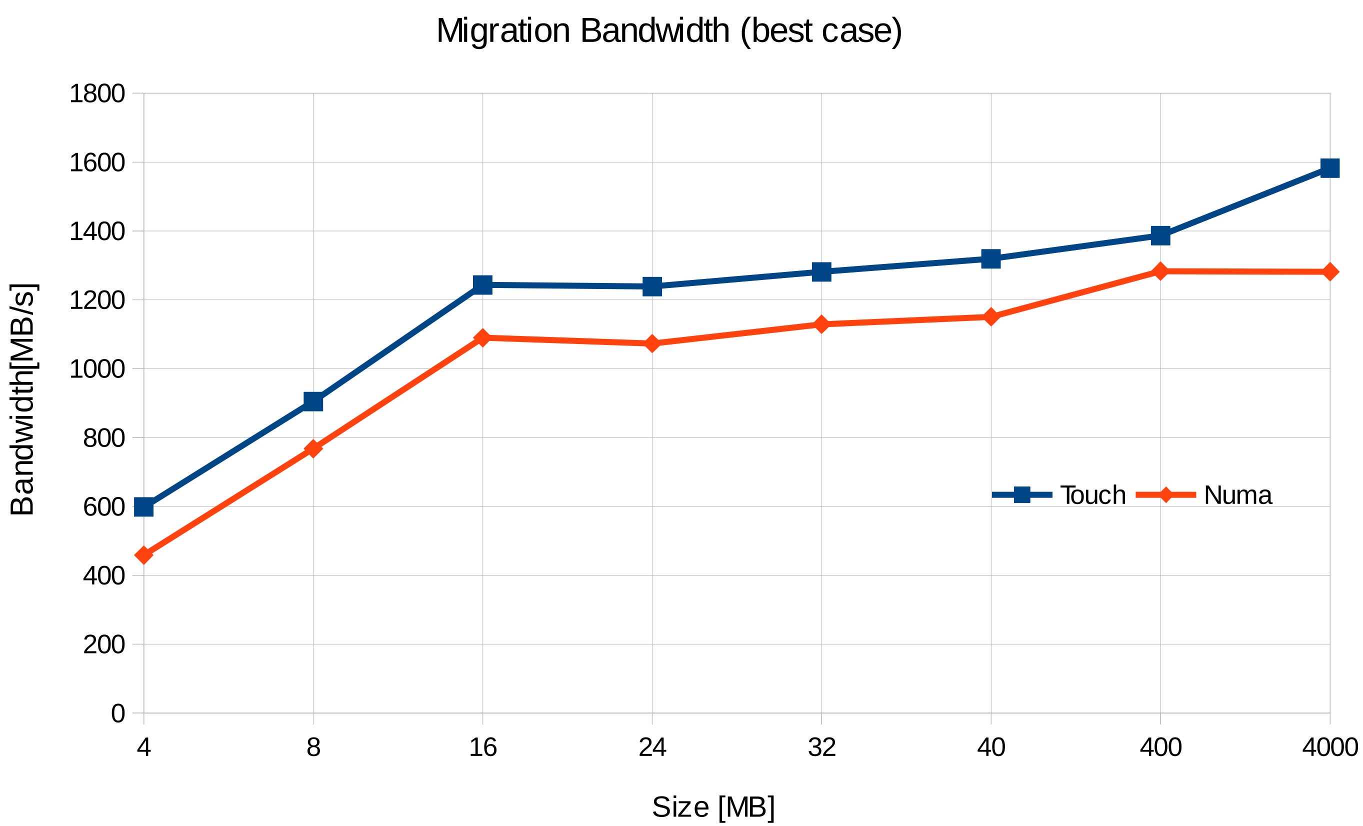
UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

```
void numavec<T>::_core_pin( int core ) {
   cpu_set_t cpuset;
   CPU_ZERO(&cpuset);
   CPU_SET( core , &cpuset);
   pthread_setaffinity_np(pthread_self(), sizeof(cpu_set_t),
&cpuset);
}
```

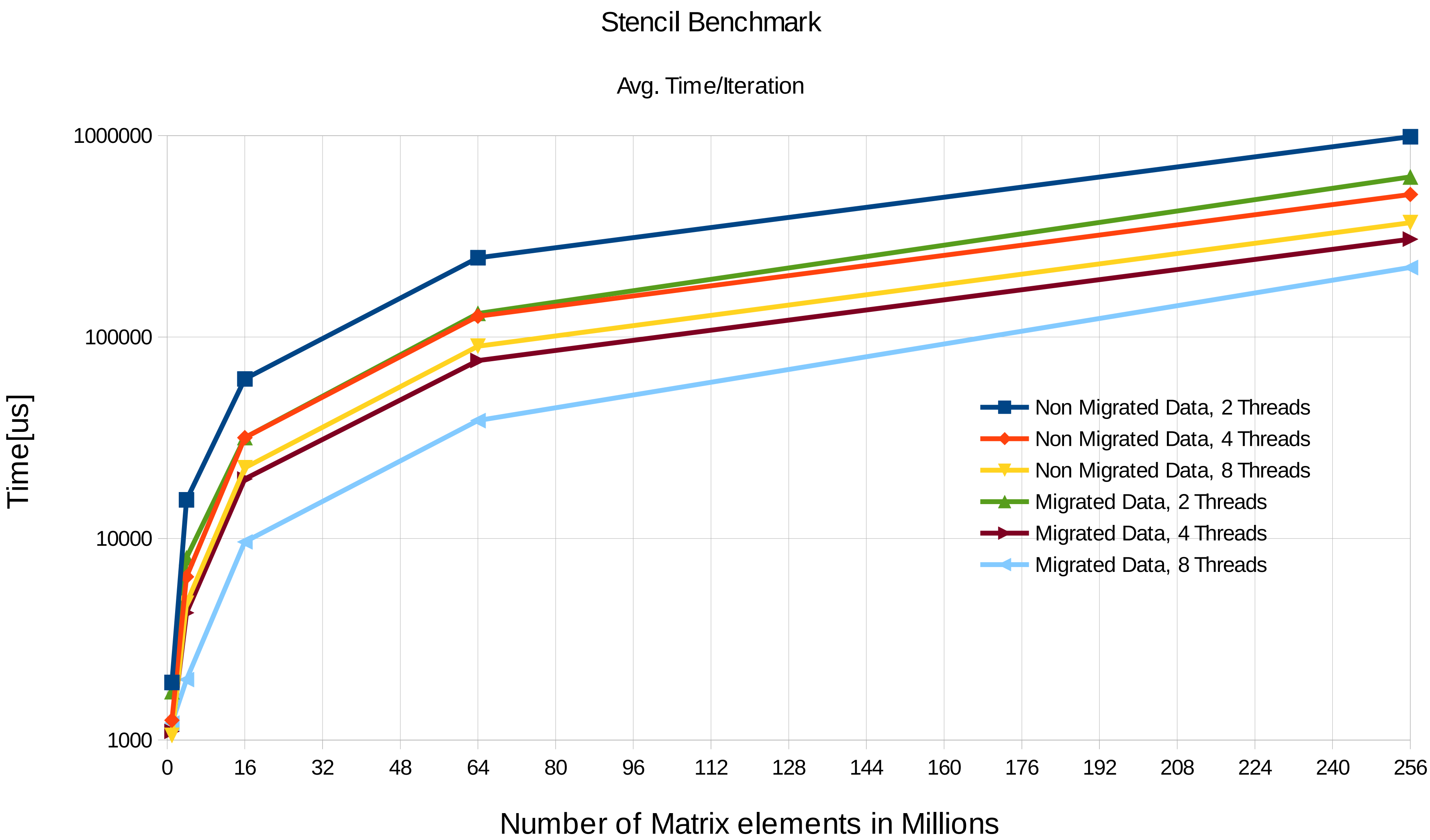
Structure

- Introduction
  - What is NUMA?
  - Motivation – Why are we doing this?
  - State of the Art – What else is out there?
- Realization – What can we do about the Problem?
  - Goals and Concepts
  - Interface
  - Implementation
- Verification – Was it all worth it?
  - Migration
  - Stencil
- Summary

Verification - Migration



Verification – Stencil



Structure

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

- Introduction
  - What is NUMA?
  - Motivation – Why are we doing this?
  - State of the Art – What else is out there?
- Realization – What can we do about the Problem?
  - Goals and Concepts
  - Interface
  - Implementation
- Verification – Was it all worth it?
  - Migration
  - Stencil
- Summary

Summary

- Achieved core and target goals
- Created easy to use container that hides NUMA complexity
- Future Goals
  - Intelligent distribution
  - Portable Version
  - Separate NUMA administration and Container administration

C++11 Features

UNIVERSITÄT HEIDELBERG
ZUKUNFT SEIT 1386

- Mutex
- Move Semantics
- Threads
- Initializer lists
- Constructor delegation