Acceleration of Blender Cycles Render Engine using Intel Xeon Phi

VŠB – Technical University of Ostrava, IT4Innovations, Czech Republic

Milan Jaroš

Presentation parts

- Blender Cycles introduction
- Algorithm for image rendering
- New OMP Device for rendering (OpenMP threads)
- New MPI Device for rendering (Message Passing Interface)
- Benchmark (Tatra T87, House, Worm)
- Live Demo

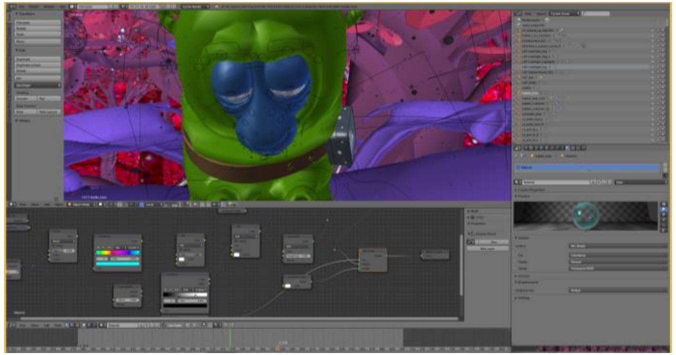
Presentation parts

- Blender Cycles introduction
- Algorithm for image rendering
- New OMP Device for rendering (OpenMP threads)
- New MPI Device for rendering (Message Passing Interface)
- Benchmark (Tatra T87, House, Worm)
- Live Demo

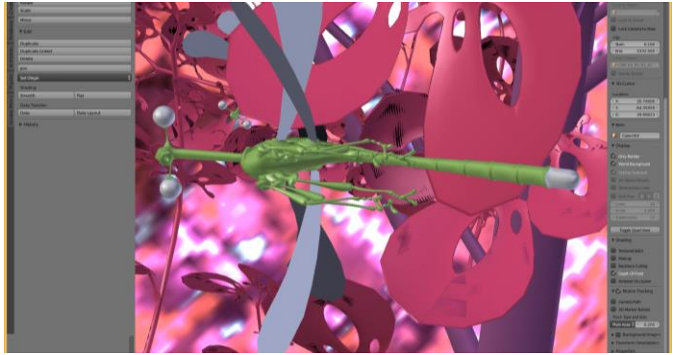
Cosmos Laundromat - First Cycle

Blender Cycles

- **Blender** is an open source 3D creation suite. It has two render engines: Blender Internal and Cycles.
- **Cycles** is a raytracing based render engine with support for interactive rendering, shading node system, and texture workflow.

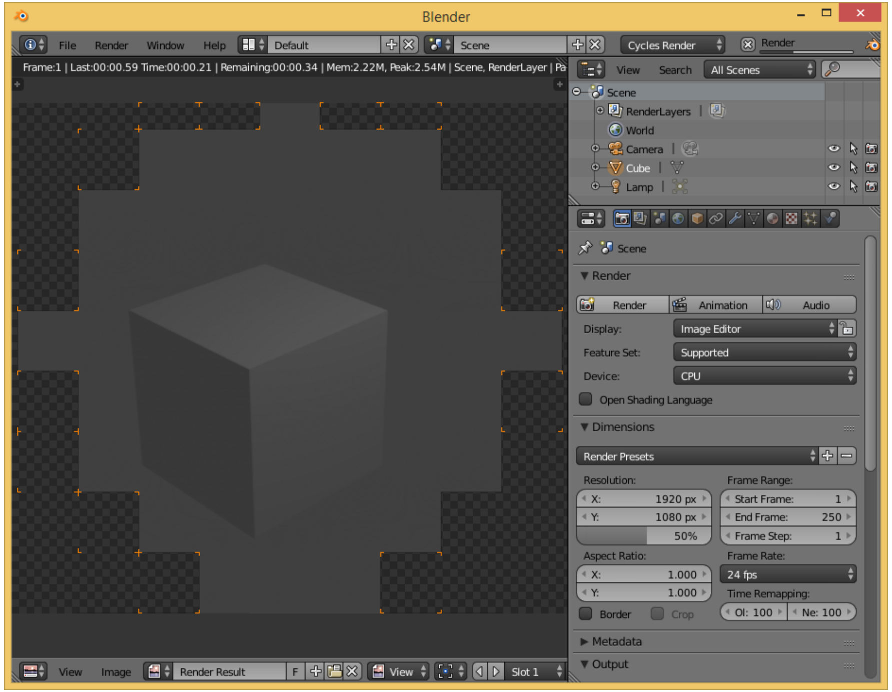


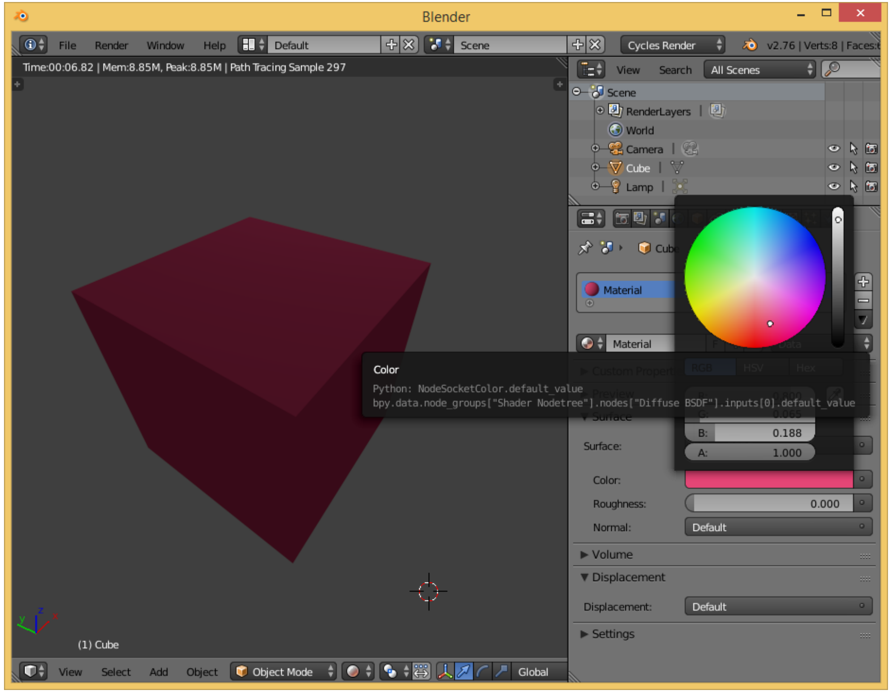
Cosmos Laundromat - First Cycle



Cosmos Laundromat - First Cycle

Difference between offline and interactive rendering

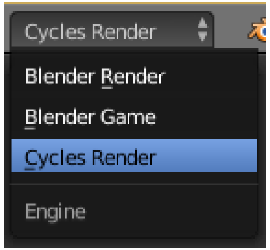


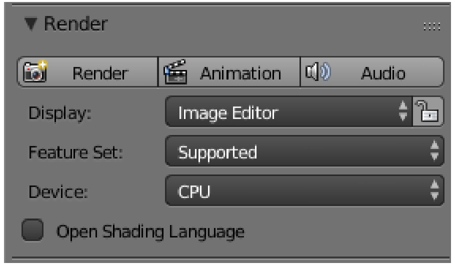


Cycles is internal plugin (Extending Python w. C++)

```
//blender/intern/cycles/blender/blender_python.cpp
static PyMethodDef methods[] = {
  //...
 { "render", render_func, METH_O, "" },
 { "bake", bake_func, METH_VARARGS, "" },
 { "draw", draw_func, METH_VARARGS, "" },
 //...
 { NULL, NULL, 0, NULL },
};

//blender/intern/cycles/blender/blender_python.cpp
static struct PyModuleDef module = {
 PyModuleDef_HEAD_INIT,
 "_cycles",
 "Blender cycles render integration",
 -1,
 methods,
 NULL, NULL, NULL, NULL
};

//blender/source/blender/python/intern/bpy_interface.c
static struct _inittab bpy_internal_modules[] = {
 { "mathutils", PyInit_mathutils },
 //...
 { "_cycles", CCL_initPython },
 //...
 { NULL, NULL }
};
```

```
//blender/intern/cycles/blender/blender_session.cpp
void BlenderSession::render() {
 //...
 BL::RenderSettings r = b_scene.render();
 //...
}

//blender/intern/cycles/blender/blender_python.cpp
static PyObject *render_func(PyObject * /*self*/, PyObject *value) {
 BlenderSession *session = (BlenderSession*)PyLong_AsVoidPtr(value);
 //...
 session->render();
 //...
 Py_RETURN_NONE;
}

//blender/intern/cycles/blender/blender_python.cpp
void *CCL_initPython() {
 PyObject *mod = PyModule_Create(&ccl::module);
 //...
 return (void*)mod;
}

//blender/intern/cycles/blender/addon/engine.py
def render(engine) :
  import _cycles
  if hasattr(engine, "session") :
    _cycles.render(engine.session)
```





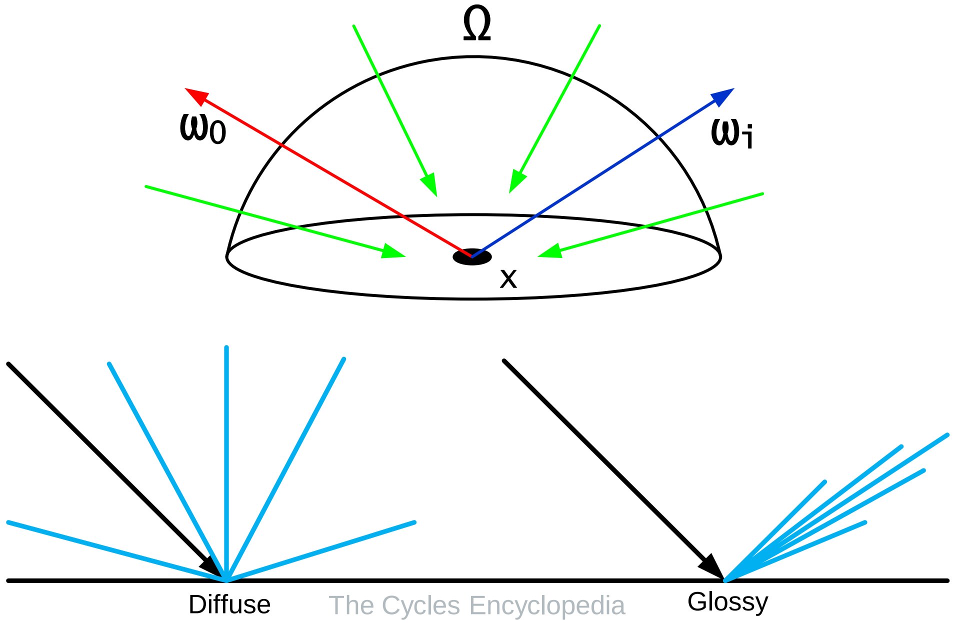
Presentation parts

- Blender Cycles introduction
- Algorithm for image rendering
- New OMP Device for rendering (OpenMP threads)
- New MPI Device for rendering (Message Passing Interface)
- Benchmark (Tatra T87, House, Worm)
- Live Demo

Rendering Equation

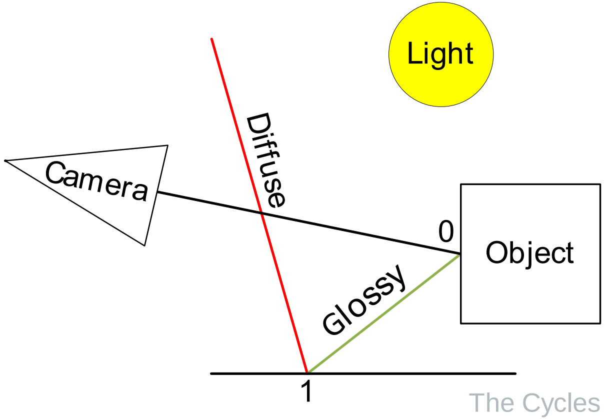
$$L_o(x,\omega_o) = L_e(x,\omega_o) + \int_{\Omega} L_i(x,\omega_i) f_r(x,\omega_i,\omega_o)(\omega_i \cdot n)\, d\omega_i$$

- ω_o is direction of outgoing ray
- ω_i is direction of incoming ray
- L_o is spectral radiance emitted by the source from point x in direction ω_o
- L_e is emitted spectral radiance from point x in direction ω_o
- Ω is the unit hemisphere in direction of normal vector n with center in x, over which we integrate
- L_i is spectral radiance coming inside to x in direction ω_i,
- $f_r(x, \omega_i, \omega_o)$ is distribution function of the image (BRDF) in point x from direction ω_i to direction ω_o,
- $\omega_i \cdot n$ is angle between ω_i and surface normal.

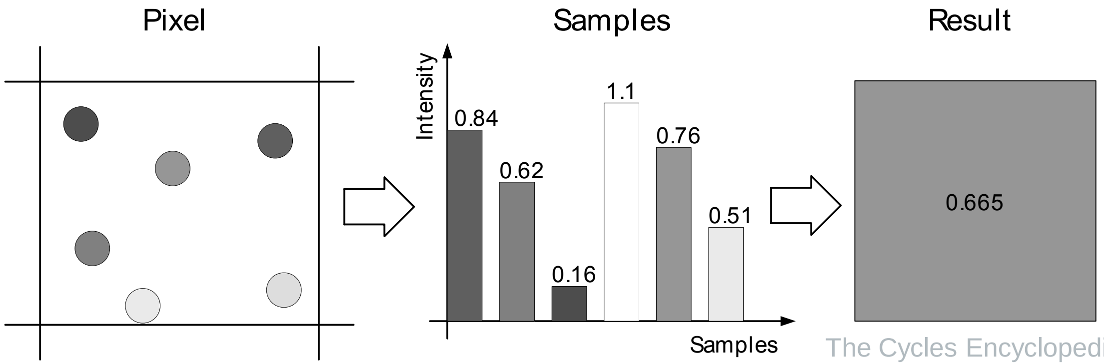


Path tracing

- For each pixel a ray is cast into a scene.
- A ray from a camera hits a glossy surface (0), then a diffuse surface (1), and it bounces into a random direction.
- The color of the ray is calculated depending on all materials of the surfaces.
- This process is repeated by the value of samples.
- The mean value of all samples is used for the color of the pixel.



The Cycles Encyclopedia

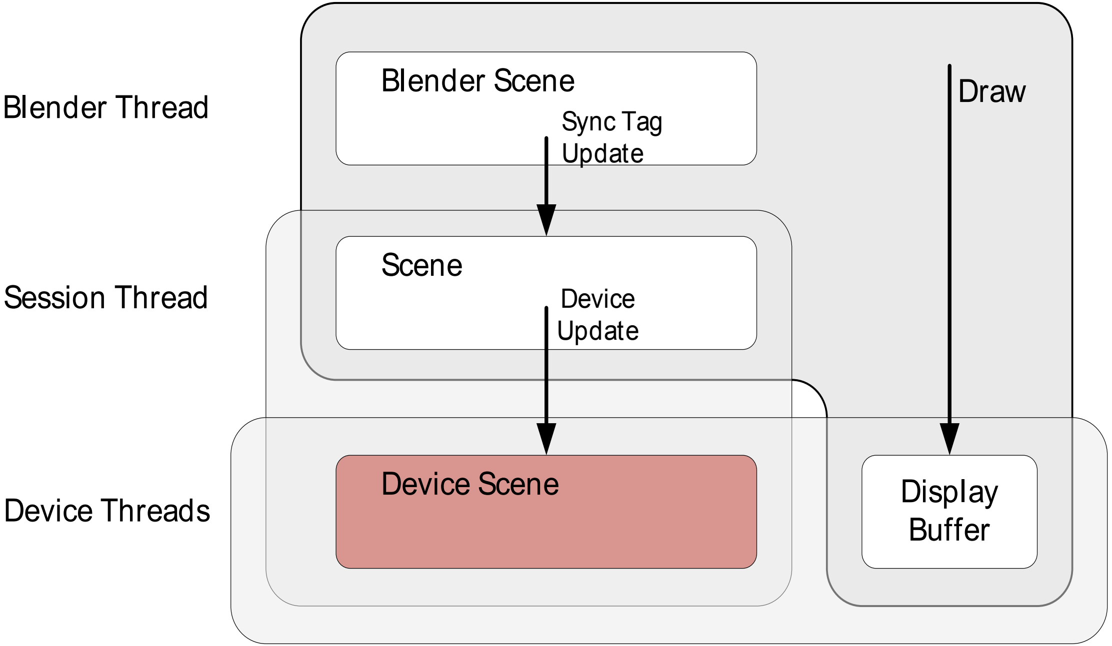


The Cycles Encyclopedia

Presentation parts

- Blender Cycles introduction
- Algorithm for image rendering
- New OMP Device for rendering (OpenMP threads)
- New MPI Device for rendering (Message Passing Interface)
- Benchmark (Tatra T87, House, Worm)

POSIX Threads in Blender



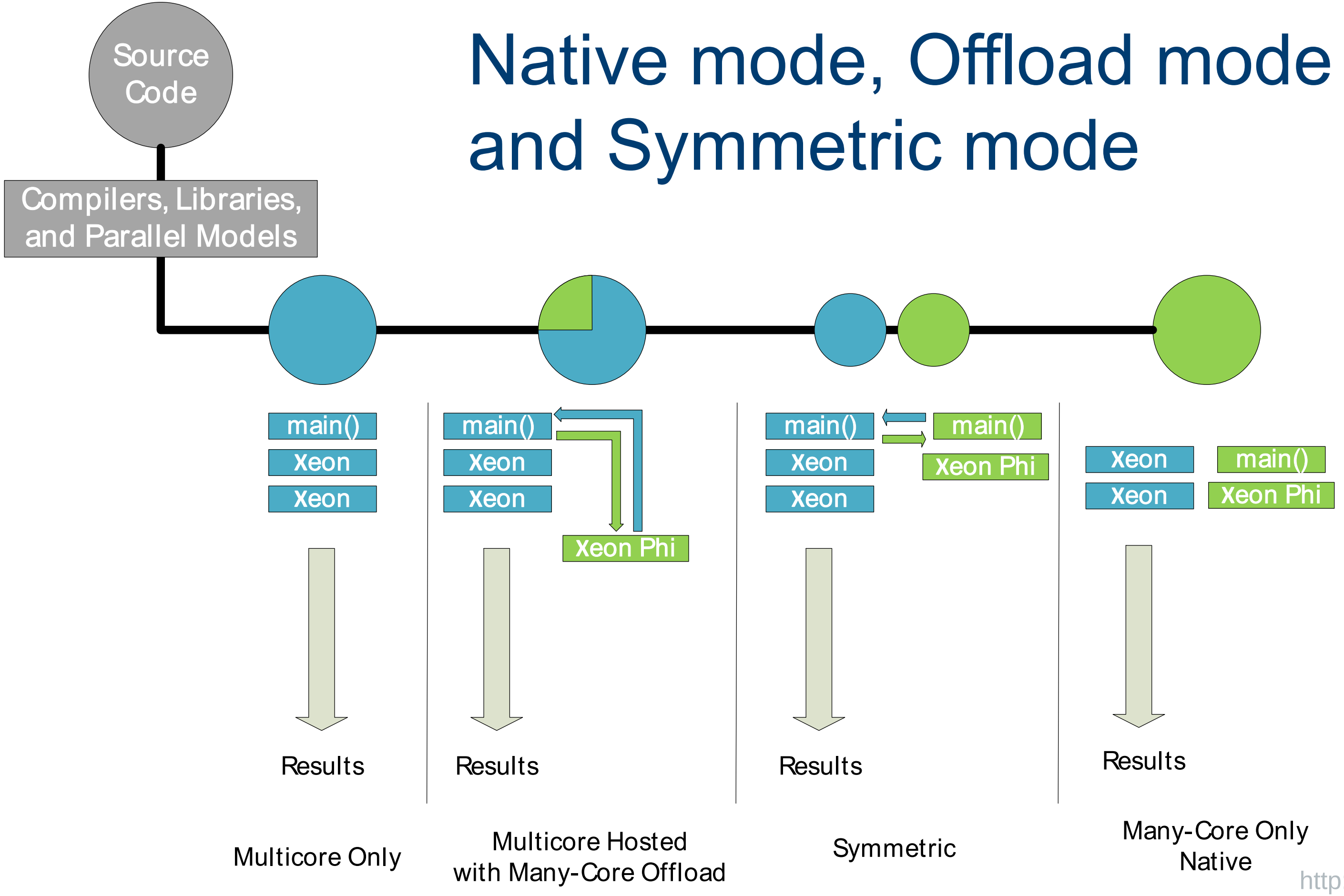
https://wiki.blender.org/index.php/Dev:Source/Render/Cycles/Threads

CyclesPhi

We have modified the kernel of the Blender Cycles rendering engine and then extended its capabilities to support the HPC environment. We call this version the CyclesPhi and it supports following technologies:

- OpenMP
- MPI
- Intel® Xeon Phi™ with Offload concept
- Intel® Xeon Phi™ with Symmetric mode
- And their combinations

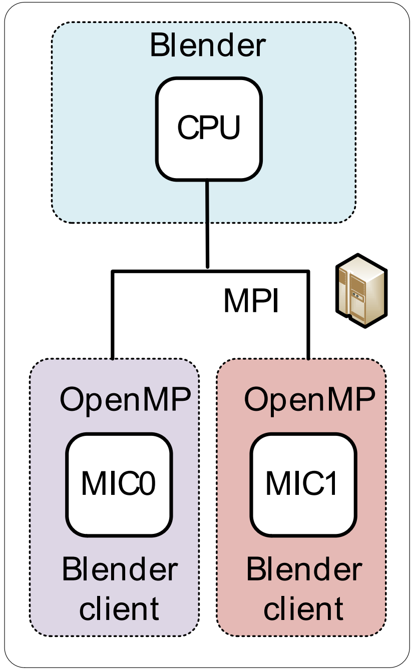
Native mode, Offload mode and Symmetric mode



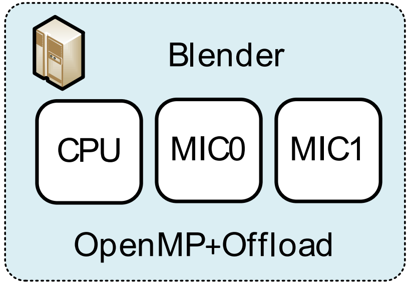
http://www.cism.ucl.ac.be

Native mode, Offload mode and Symmetric mode

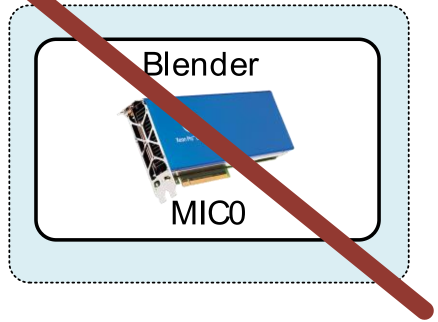
Symmetric mode



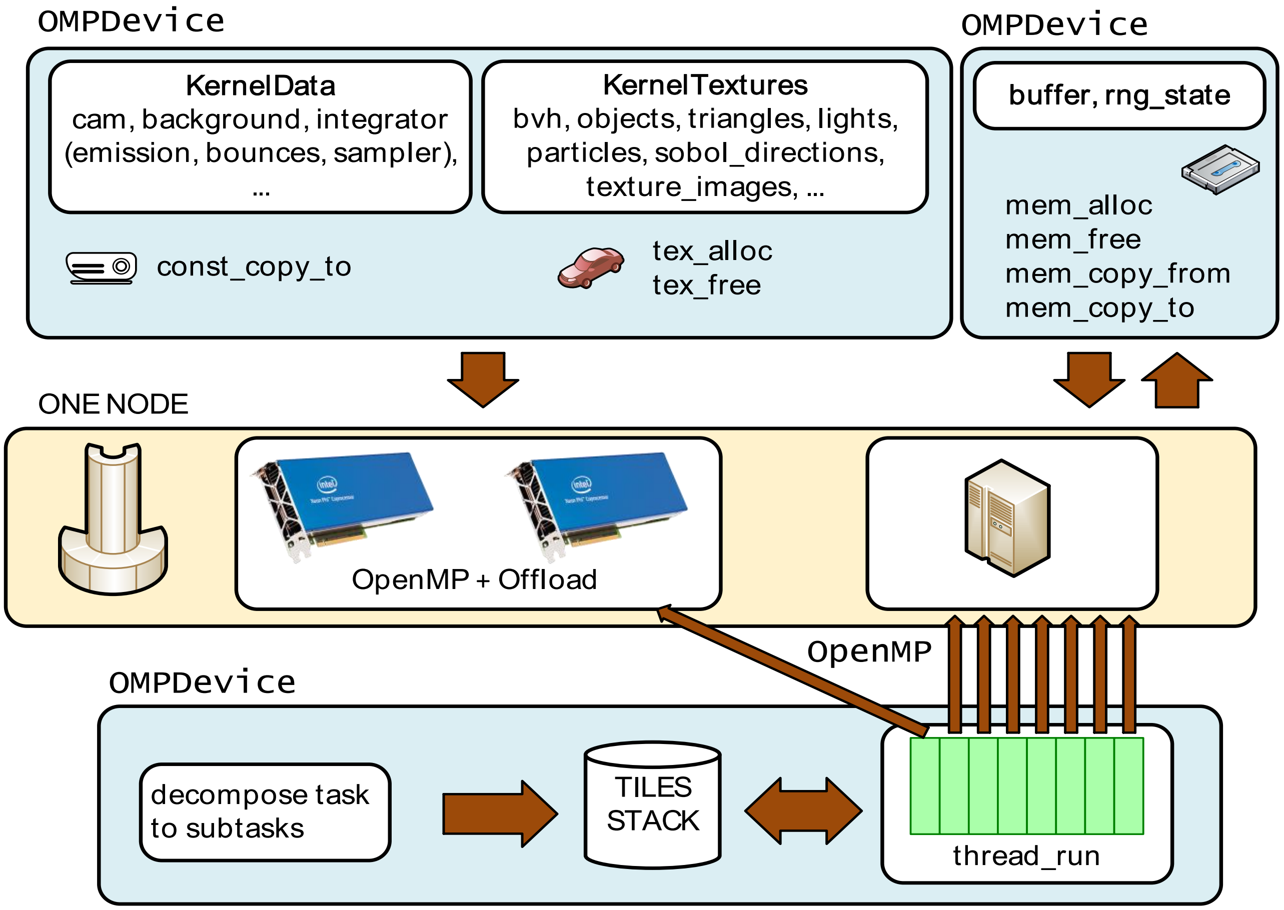
Offload mode



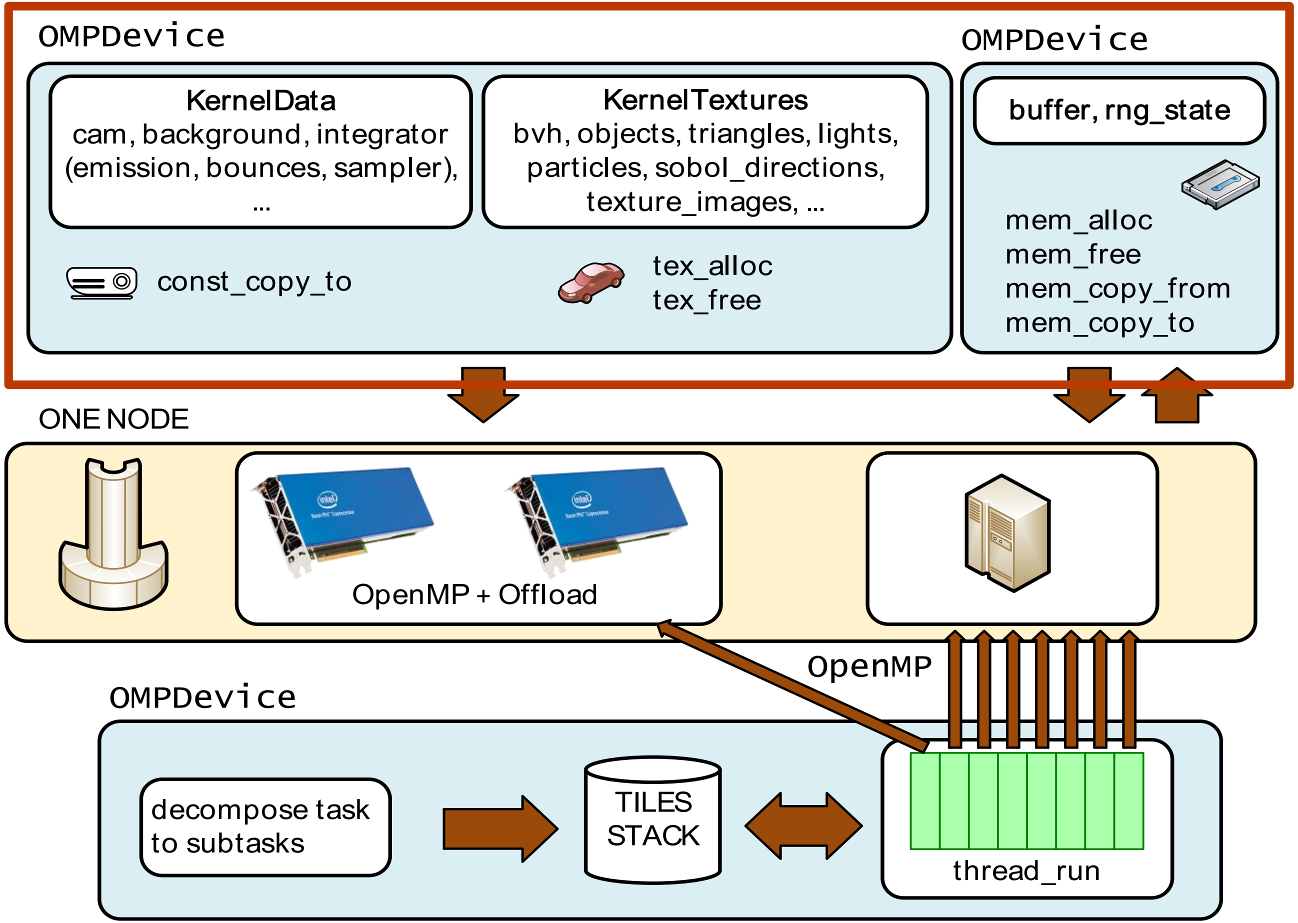
Native mode



Parallelization for MIC using OpenMP and Offload



Parallelization for MIC using OpenMP and Offload

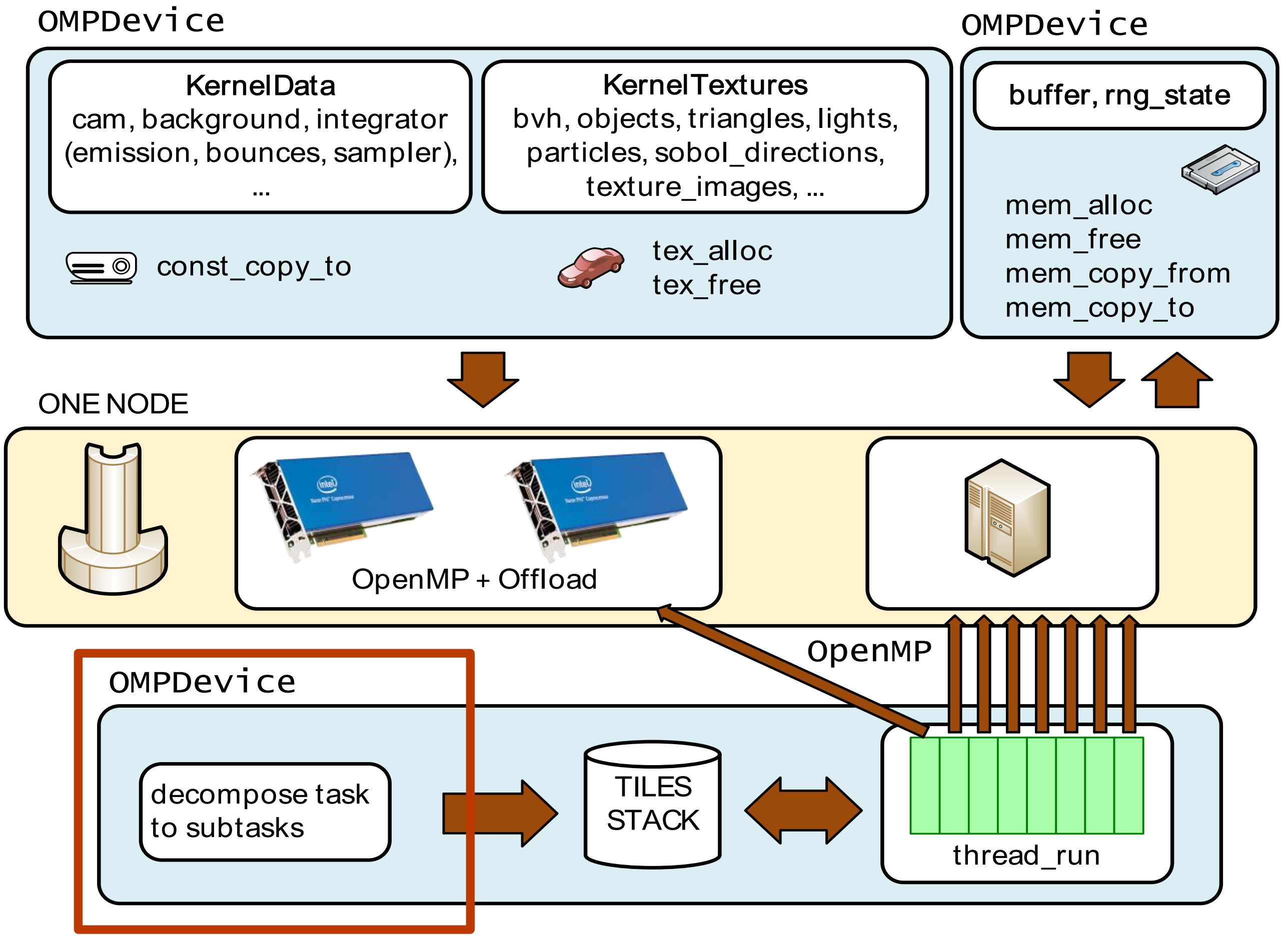


Parallelization for MIC using OpenMP and Offload

```
//blender/intern/cycles/kernel/kernels/mic/kernel_mic.cpp
#define ALLOC alloc_if(1) free_if(0)
#define FREE alloc_if(0) free_if(1)
#define REUSE alloc_if(0) free_if(0)
#define ONE_USE
device_ptr mic_alloc_kg(int numDevice) {
    device_ptr kg_bin;
    #pragma offload target(mic:numDevice) out(kg_bin)
     {
          KernelGlobals *kg = new KernelGlobals();
          kg_bin = (device_ptr) kg;
     }
    return (device_ptr) kg_bin;
}
void mic_free_kg(int numDevice, device_ptr kg_bin) {
      #pragma offload target(mic:numDevice) in(kg_bin)
       {
          KernelGlobals *kg = (KernelGlobals *) kg_bin;
          delete kg;
} }
void mic_const_copy(int numDevice, /*…*/)  {
  #pragma offload target(mic:numDevice) \\
  in(host_bin:length(size) ONE_USE) in(kg_bin) in(size)
  {
     KernelGlobals *kg = (KernelGlobals *)kg_bin;
     memcpy(&kg->__data, host_bin, size);
     kg->__data_size = size;
} }
```

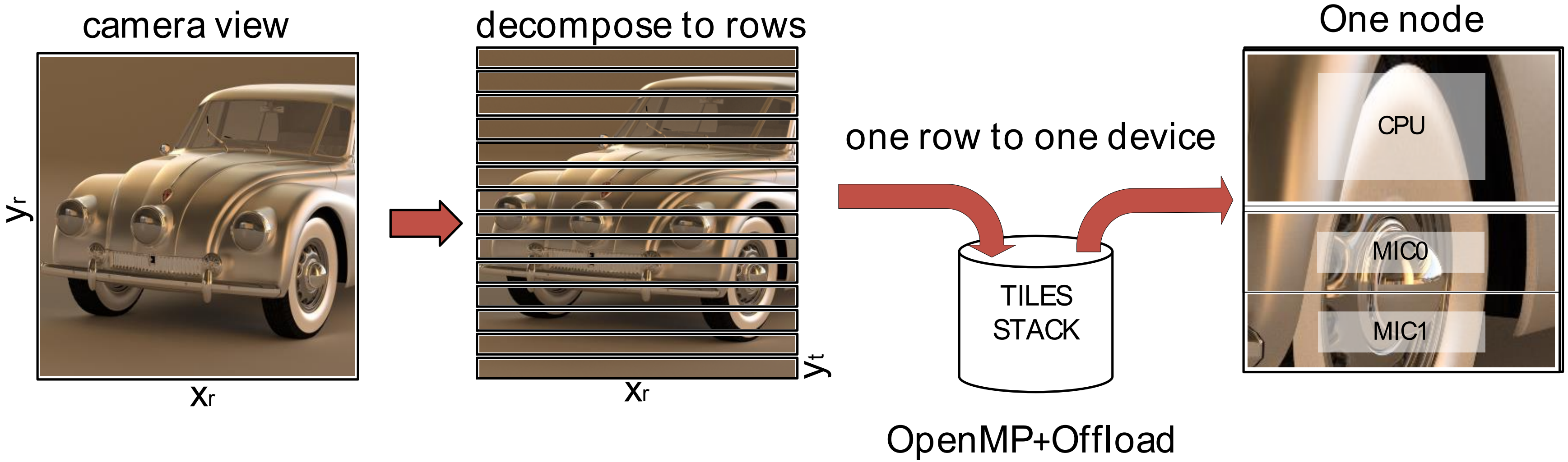
```
void mic_mem_alloc(int numDevice, char *mem, size_t memSize) {
  #pragma offload target(mic:numDevice) in(mem:length(memSize) ALLOC)
}
void mic_mem_copy_to(int numDevice, char *mem, size_t memSize, char*
signal_value) {
        if (signal_value == NULL)  {
  #pragma offload target(mic:numDevice) in(mem:length(memSize) REUSE)
        }  else  {
  #pragma offload_transfer target(mic:numDevice) in(mem:length(memSize) REUSE)
signal(signal_value)
        }
}
void mic_mem_copy_from(int numDevice, char *mem, size_t offset, size_t memSize,
char* signal_value) {
        if (signal_value == NULL)
        {
  #pragma offload target(mic:numDevice) out(mem[offset:memSize]: REUSE)
        }
        else
        {
  #pragma offload_transfer target(mic:numDevice) out(mem[offset:memSize]: REUSE)
signal(signal_value)
        }
}
void mic_mem_free(int numDevice, char *mem, size_t memSize) {
  #pragma offload target(mic:numDevice) in(mem:length(0) FREE)
}
```

Parallelization for MIC using OpenMP and Offload

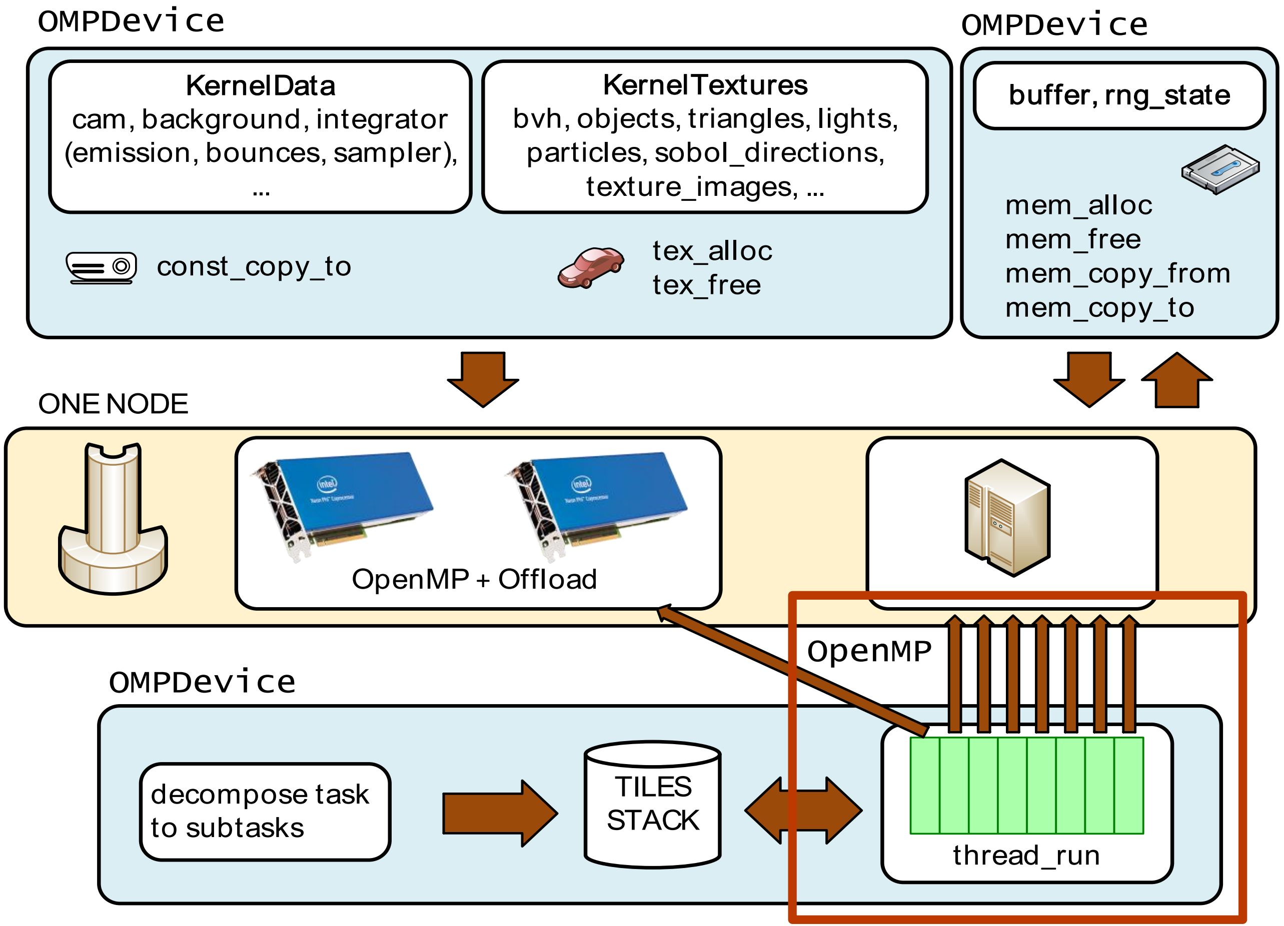


Parallelization for MIC using OpenMP and Offload

The synthesized image with resolution x(r) × y(r) is decomposed to rows (y(t) = 1). In our cases, there are three devices: CPU (24 cores), Intel Xeon Phi / MIC (61+61 cores). One device reads the stack and gets one row. The load balancing is provided by the stack.



Parallelization for MIC using OpenMP and Offload



Parallelization for MIC using OpenMP and Offload

```
//blender/intern/cycles/kernel/kernels/mic/kernel_mic.cpp
void mic_path_trace(int numDevice, /*...*/)
{
#pragma offload target(mic:numDevice) \
        in(buffer_bin : length(0) REUSE) \
        in(rng_state_bin : length(0) REUSE) \
        in(sample_finished_mic : length(0) REUSE) \
        in(reqFinished_mic : length(0) REUSE) \
        in(rgba_byte_bin : length(0) REUSE) \
        in(kg_bin) in(start_sample) in(end_sample) \
        in(tile_x) in(tile_y) in(offset) in(stride) \
        in(tile_h) in(tile_w) in(nprocs_cpu) \
        signal(signal_value)
        {
#pragma omp parallel for num_threads(nprocs_cpu) schedule(dynamic, 1)
  for (int i = 0; i < size; i++)
  {
   int y = i / tile_w;
   int x = i - y * tile_w;

   for (int sample = start_sample; sample < end_sample; sample++)
   {
     kernel_path_trace((KernelGlobals *)kg_bin, /*…*/);
} } }
```

```
//blender/intern/cycles/device/device_omp.cpp
omp_set_nested(1);
#pragma omp parallel num_threads(2)          {
#pragma omp single nowait                  {
#pragma omp task {
            while (reqFinished == 0) {
              #pragma omp flush
              if (omp_path_trace_req != 0)  {
                cpu_path_trace((KernelGlobals *)kg_bin, /*...*/);
                omp_path_trace_req = 0;
              }
              usleep(100);
            } }
#pragma omp task {
            while (true) {
              for (int dev = 0; dev < num_devices_cpu_mics; dev++) {
                if (dev > 0)
                  mic_mem_copy_from(dev - 1, (char*) buffer, /*...*/);
                if (sample_finished_devices[dev] == end_sample) {
                  if (dev == 0) omp_path_trace_req = 1;
                  else  mic_path_trace(dev - 1, /*...*/);
              }  }
              task.update_progress(&tile);
              //...
            }  }  }
#pragma omp taskwait   }
```

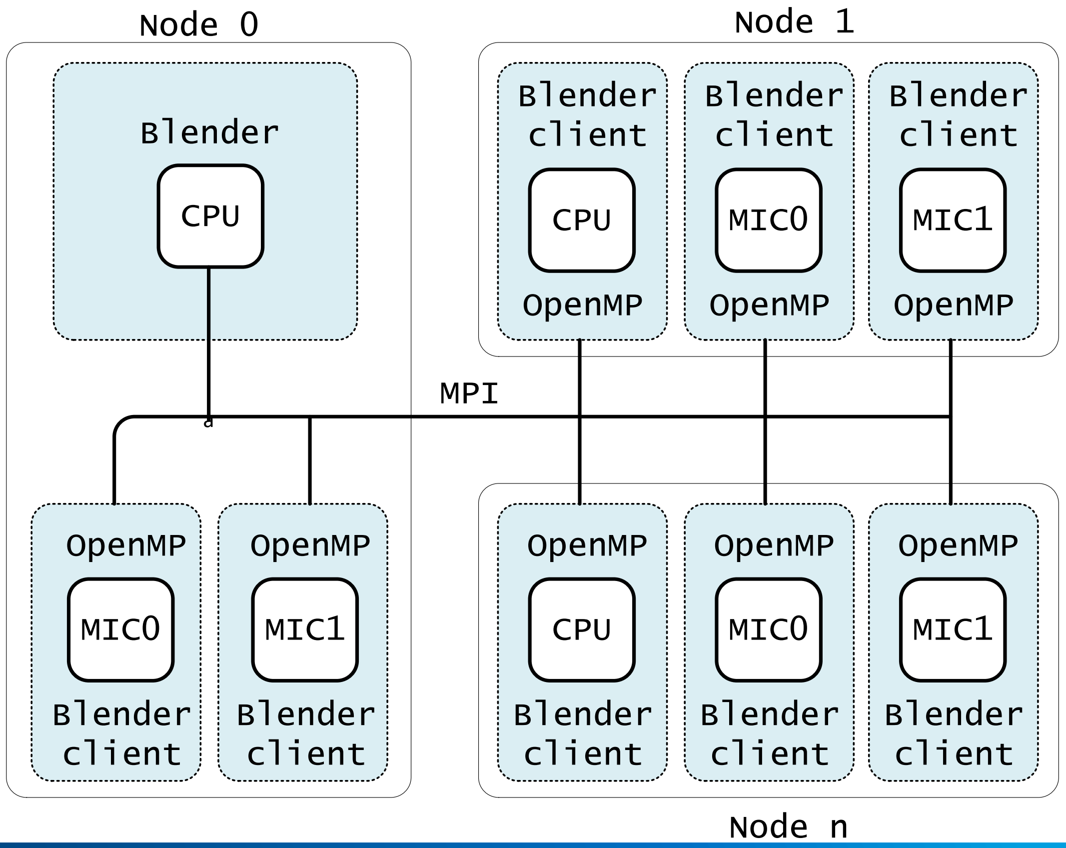
Presentation parts

- Blender Cycles introduction
- Algorithm for image rendering
- New OMP Device for rendering (OpenMP threads)
- New MPI Device for rendering (Message Passing Interface)
- Benchmark (Tatra T87, House, Worm)
- Live Demo

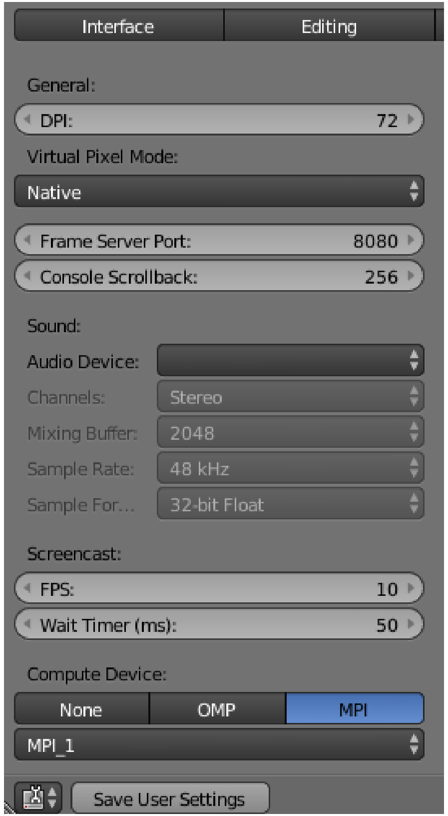
Rendering using OpenMP, Symmetric mode and MPI

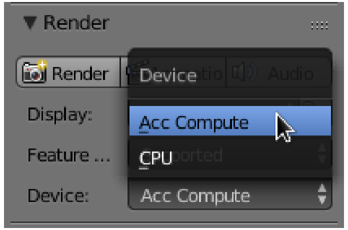
build flags:

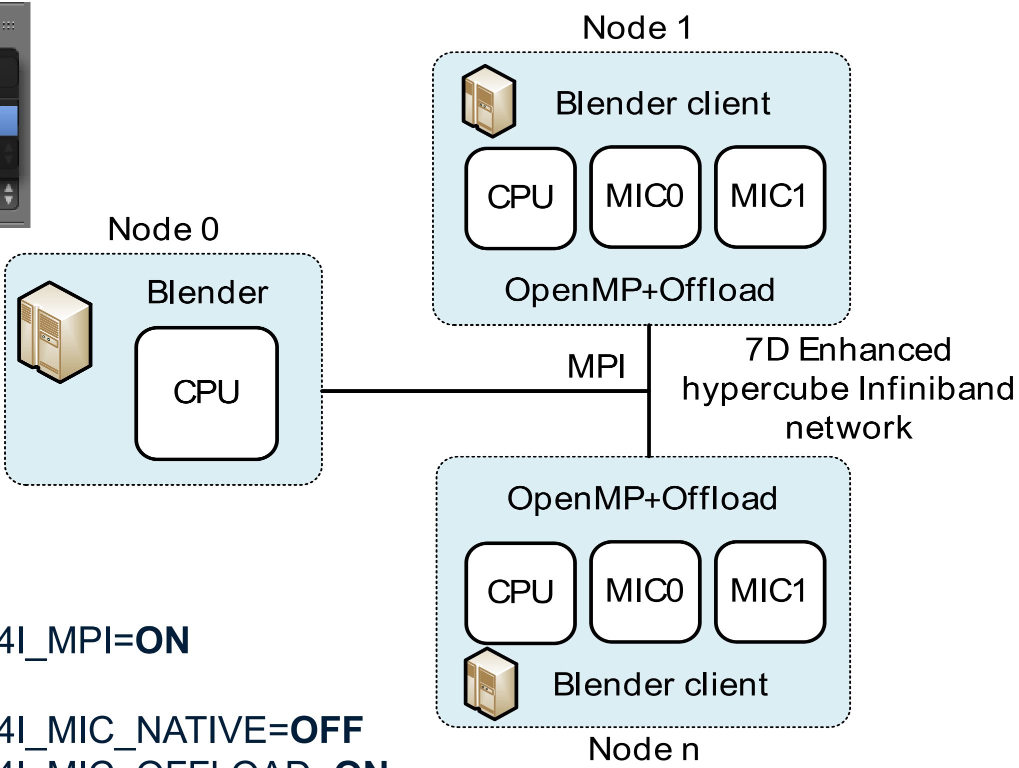
- blender
  - WITH_IT4I_MPI=**ON**
- client-cpu
  - WITH_IT4I_MIC_NATIVE=**OFF**
  - WITH_IT4I_MIC_OFFLOAD=**OFF**
- client-mic
  - WITH_IT4I_MIC_NATIVE=**ON**
  - WITH_IT4I_MIC_OFFLOAD=**OFF**



Rendering using OpenMP, Offload and MPI



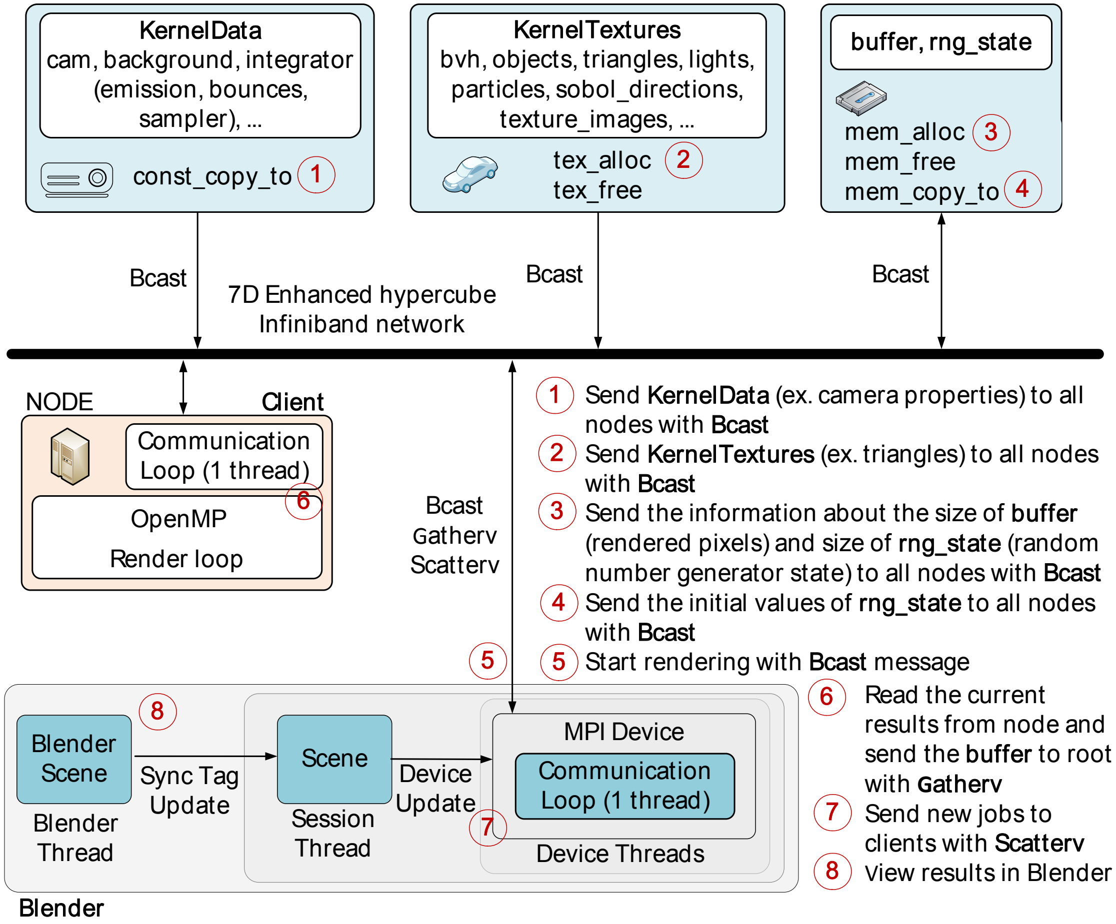




build flags:

- blender
  - WITH_IT4I_MPI=**ON**
- client
  - WITH_IT4I_MIC_NATIVE=**OFF**
  - WITH_IT4I_MIC_OFFLOAD=**ON**

Offline rendering using OpenMP and MPI



1. Send **KernelData** (ex. camera properties) to all nodes with **Bcast**
2. Send **KernelTextures** (ex. triangles) to all nodes with **Bcast**
3. Send the information about the size of **buffer** (rendered pixels) and size of **rng_state** (random number generator state) to all nodes with **Bcast**
4. Send the initial values of **rng_state** to all nodes with **Bcast**
5. Start rendering with **Bcast** message
6. Read the current results from node and send the **buffer** to root with **Gatherv**
7. Send new jobs to clients with **Scatterv**
8. View results in Blender

Offline rendering using OpenMP and MPI

```
while (true) {
  //receive sample from client
  MPI_Gatherv(/*...*/);
  //receive the computed row from client
  MPI_Gatherv(/*...*/);
  //receive rendered row from client
  MPI_Gatherv(/*...*/);
  //check the work of client and generate new job
  int min_count = end_sample;
  for (int i = 0; i < dev_count; i++) {
    if (min_count > sample_finished[i])
      min_count = sample_finished[i];
    if (sample_finished[i] == end_sample)
       reqJob[i] = tile_y_node++;
    else
       reqJob[i] = -1;
  }
  //refresh view
  task.update_progress(&tile);
  //send job to client
  MPI_Scatterv(/*...*/);
  //check all finished job and quit
  if (reqFinished != 0) break;
}
```

```
omp_set_nested(1); //need for omp_parallel in omp_parallel
#pragma omp parallel num_threads(2) {
#pragma omp single nowait {
#pragma omp task {
   while (reqFinished == 0) {
      #pragma omp flush
      if (omp_path_trace_req != 0) {
        #pragma omp parallel for schedule(dynamic, 1)
        for (int i = 0; i < size; i++) {
          /*...*/
          kernel_path_trace(/*...*/);
        }
        omp_path_trace_req = 0;
     }
     usleep(100);
}  }
#pragma omp task {
   while (true) {
MPI_Gatherv(/*...*/); //send sample to root
MPI_Gatherv(/*...*/); //send the computed row to root
MPI_Gatherv(/*...*/); //send rendered row to root
MPI_Scatterv(/*...*/); //receive job to client
if (reqJob >= 0) omp_path_trace_req = 1; //check/start new job
if (reqFinished != 0) break;
   } } } #pragma omp taskwait }
```

Presentation parts

- Blender Cycles introduction
- Algorithm for image rendering
- New OMP Device for rendering (OpenMP threads)
- New MPI Device for rendering (Message Passing Interface)
- Benchmark (Tatra T87, House, Worm)
- Live Demo

Benchmark (Tatra T87, House, Worm)

- **The benchmark** was run on one computing node of the Salomon supercomputer equipped with two Intel Xeon E5-2680v3 CPUs and two Intel Xeon Phi 7120P.
- **GPU** test was run on two NVIDIA GeForce GTX 970.

Tatra T87

Tatra T87 by David Cloete

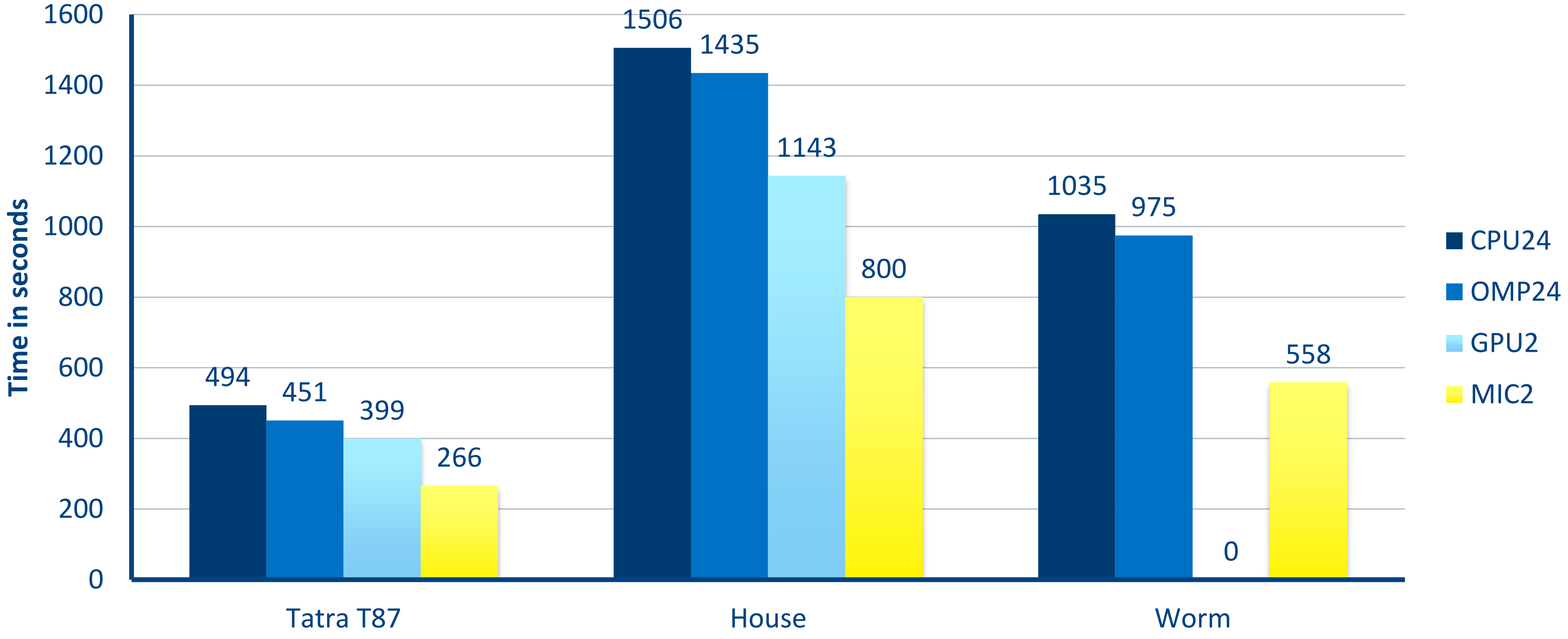
House

Pabellon Barcelona by Claudio Andres

Worm

Cosmos Laundromat - First Cycle

Performance comparison MIC with other devices

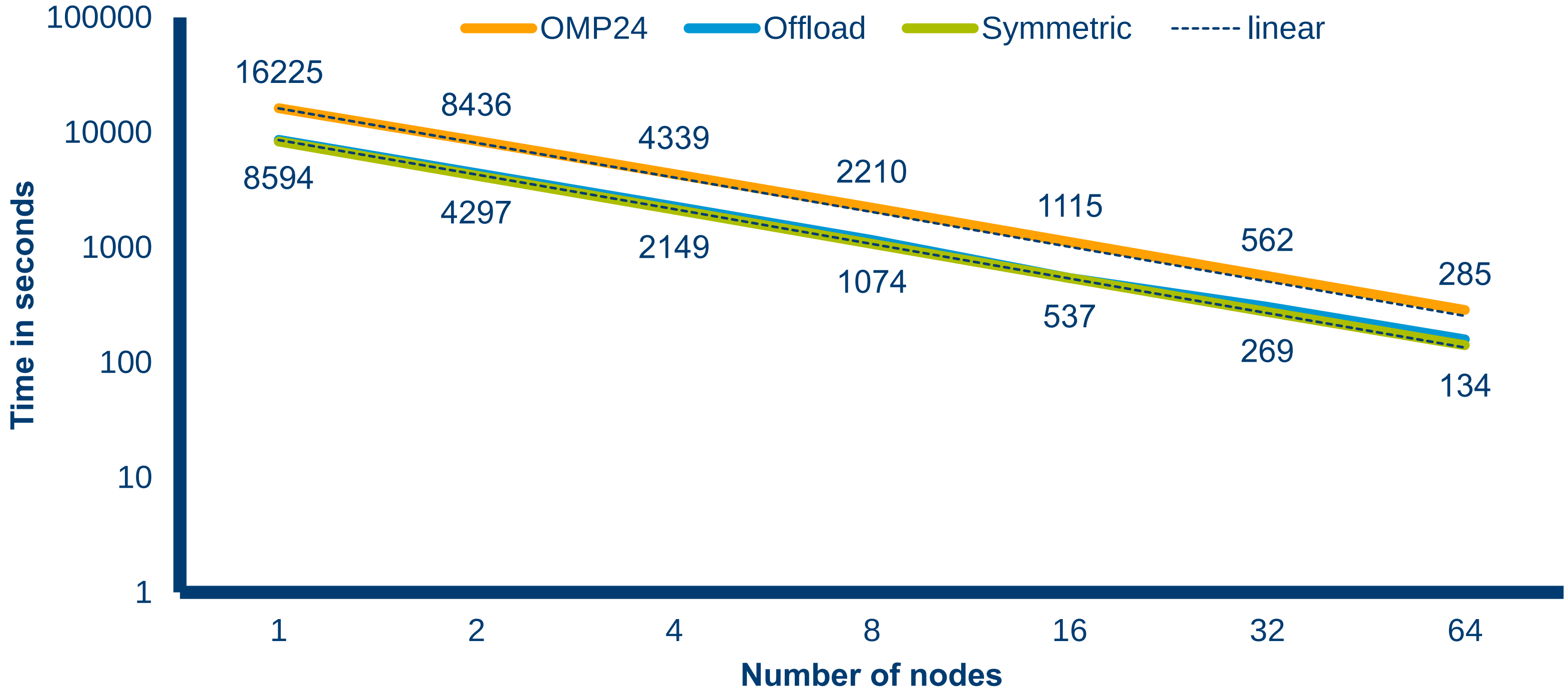


Benchmark Worm: Strong Scalability MPI Test (offline)

- **The benchmark** was run on 64 computing nodes of the Salomon supercomputer equipped with two Intel Xeon E5-2680v3 CPUs and two Intel Xeon Phi 7120P.
- **Worm scene** has 13.2 million triangles.
- Resolution: 4096x2048, Samples: 1024

Cosmos Laundromat - First Cycle

Benchmark Worm: Strong Scalability MPI Test (offline)

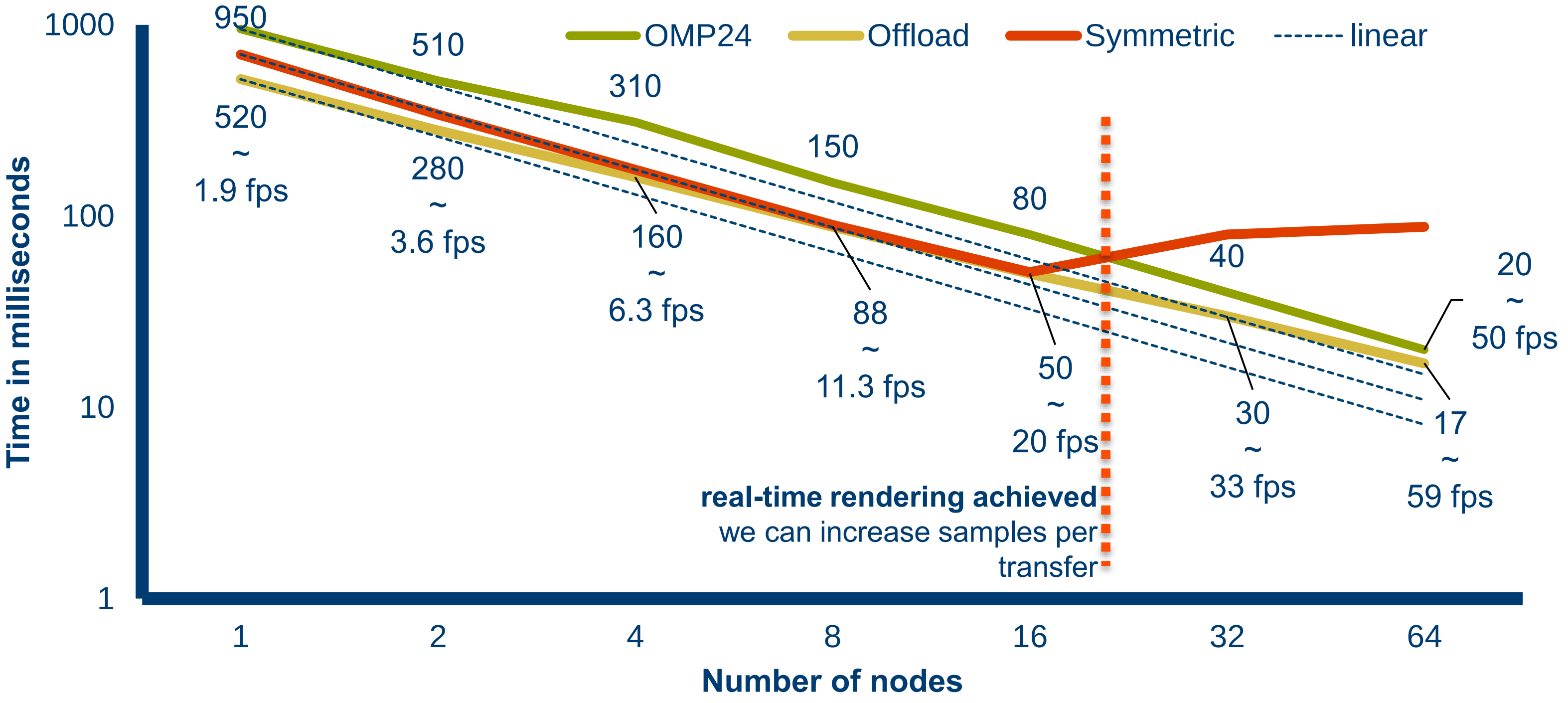


Benchmark Tatra T87: Strong Scalability MPI Test (interactive)

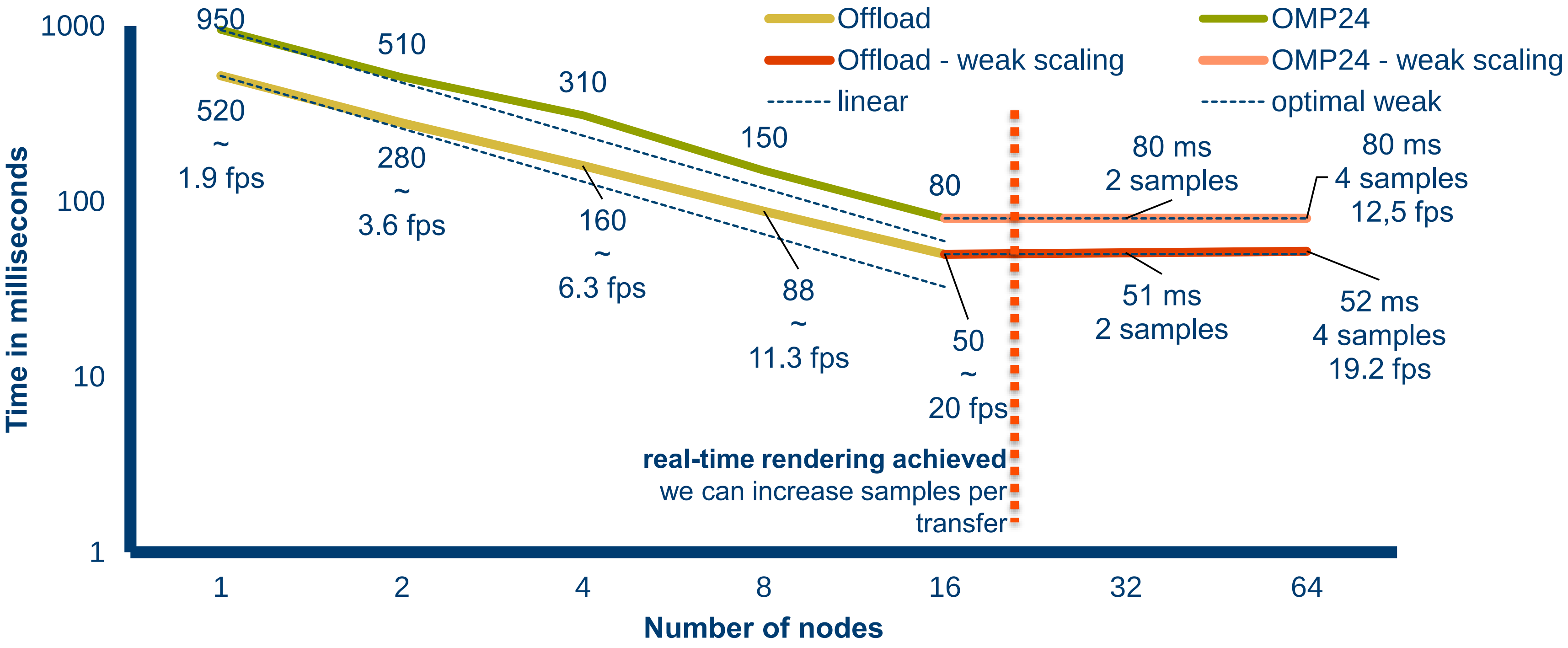
- **The benchmark** was run on 64 computing nodes of the Salomon supercomputer equipped with two Intel Xeon E5-2680v3 CPUs
- **Tatra T87** has 1.2 million triangles and uses the HDRI lighting.
- Resolution: 1920x1080, Samples: 1

Tatra T87 by David Cloete

Benchmark Tatra T87: Strong Scalability MPI Test (interactive - 1 sample)



Benchmark Tatra T87: Strong Scalability MPI Test (interactive - 1 sample)



Presentation parts

- Blender Cycles introduction
- Algorithm for image rendering
- New OMP Device for rendering (OpenMP threads)
- New MPI Device for rendering (Message Passing Interface)
- Benchmark (Tatra T87, House, Worm)
- Live Demo

References

- Jaros Milan, et al.: Acceleration of Blender Cycles Path-Tracing Engine Using Intel Many Integrated Core Architecture, CISIM 2015, Warsaw, Poland, p. 86-97, September 2015
- Frederik Steinmetz, Gottfriend Hofmann: The Cycles Encyclopedia
- https://wiki.blender.org
- https://www.youtube.com/watch?v=Y-rmzh0PI3c
- https://cloud.blender.org/blog/cycles-turbocharged-how-we-made-rendering-10x-faster