FFMK: BUILDING AN EXASCALE OPERATING SYSTEM

Amnon Barak  Hebrew University Jerusalem (HUJI)
Hermann Härtig  TU Dresden, Operating Systems Group (TUDOS)
Wolfgang E. Nagel  TU Dresden, Center for Information Services and HPC (ZIH)
Alexander Reinefeld  Konrad-Zuse-Zentrum für Informationstechnik Berlin (ZIB)

CARSTEN WEINHOLD, TU DRESDEN
CHALLENGES

- Heterogeneity (cores, memory, …)
- Power / dark silicon
- Increased failure rates
- „OS Noise“
- Dynamic platforms & applications
The dynamic platform management consists of two basic components: monitoring and decision-making. To achieve the scalability and resilience required for Exascale computing, we require an application model that is more flexible than the coarse-grained and static division of resources. For example, a core may run multiple tasks (one after each other) that communicate with each other by preempting at blocking communication calls—a principle called overdecomposition (see Figure 4).

The units of decomposition are migratable, allowing the dissemination of the monitoring component (see Section 4.2) and making decisions scale systems. We decided to use gossip algorithms for all cross-node information and decision making. To achieve the scalability and resilience required for Exascale computing, we require an application model that is more flexible than the coarse-grained and static division of resources. For example, a core may run multiple tasks (one after each other) that communicate with each other by preempting at blocking communication calls—a principle called overdecomposition (see Figure 4).

The dynamic platform management consists of two basic components: monitoring and decision-making. To achieve the scalability and resilience required for Exascale computing, we require an application model that is more flexible than the coarse-grained and static division of resources. For example, a core may run multiple tasks (one after each other) that communicate with each other by preempting at blocking communication calls—a principle called overdecomposition (see Figure 4).

The dynamic platform management consists of two basic components: monitoring and decision-making. To achieve the scalability and resilience required for Exascale computing, we require an application model that is more flexible than the coarse-grained and static division of resources. For example, a core may run multiple tasks (one after each other) that communicate with each other by preempting at blocking communication calls—a principle called overdecomposition (see Figure 4).

The dynamic platform management consists of two basic components: monitoring and decision-making. To achieve the scalability and resilience required for Exascale computing, we require an application model that is more flexible than the coarse-grained and static division of resources. For example, a core may run multiple tasks (one after each other) that communicate with each other by preempting at blocking communication calls—a principle called overdecomposition (see Figure 4).
Architecture

Three main results:
- Runs on a real system
- Execution model + noise
- Oversubscription / load balancing

Outlook: Application hints

Summary
FFMK: Building an Exascale Operating System
Minimal amount of cycles for work
Ideal: zero extra cycles

+ 0 cycles
Real-World HPC Linux

+450,000 cycles ≈ 10%
• Use **Linux**, but move it out of the way

• Put state-of-the-art L4 **microkernel** in control
Decoupled Linux thread

+60 cycles
Decoupled Linux thread

+4 cycles
Ideal behavior

![Ideal behavior diagram](image)

**Process ID**
- 1023
- 767
- 512
- 256
- 0

**Timesteps**
- 10
- 20
- 30
- 40
- 50

**Computation time (fraction)**
- 1
- 0.9
- 0.8
- 0.7
- 0.6
- 0.5
- 0.4
- 0.3
- 0.2
- 0.1
Application: CP2K

LOAD IMBALANCES
Application: COSMO-SPECS+FD4

Load Imbalances

Process ID vs. Timesteps

Computation time (fraction)
Application: COSMO-SPECS+FD4

Polling (busy waiting)

Polling

Blocking

# of MPI Processes per Node

# of Nodes

FFMK: Building an Exascale Operating System
Application: COSMO-SPECS+FD4

**RUN TIME**

- **Unbalanced, no HT**
- **Balanced (baseline), no HT**

<table>
<thead>
<tr>
<th># of Nodes</th>
<th># of MPI Processes per Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>12 / 112</td>
<td>12 / 112</td>
</tr>
<tr>
<td>24 / 112</td>
<td>24 / 112</td>
</tr>
<tr>
<td>48 / 112</td>
<td>48 / 112</td>
</tr>
</tbody>
</table>

FFMK: Building an Exascale Operating System
Application: COSMO-SPECS+FD4

Unbalanced, no HT

# of MPI Processes per Node
12 / 112
24 / 56
48 / 28

# of Nodes
Application: COSMO-SPECS+FD4

Unbalanced, no HT

- 0 h
- 40 h
- 80 h
- 120 h
- 160 h
- 200 h

# of MPI Processes per Node

- 12 / 112
- 24 / 56
- 48 / 28

# of Nodes
Application: CP2K

Core Hours

- Unbalanced, no HT

- # of MPI Processes per Node: 12 / 112, 24 / 56, 48 / 28
- # of Nodes
Oversubscription does work
Better utilization / throughput
„Hyper Threading“ even better
No migration yet …
… but Gossip
- Spread load+health info among nodes
- Analytic model ~ simulation ~ emulation
- Negligible overhead (64–256 ms intervals)
- Good quality of information (2–3 s old)
- Fault tolerant (simulated for up to 32 of 1024 nodes failing)


APPLICATION HINTS

Application

Runtime

Operating System
Resource usage:
- CPU cycles [energy?]
- Cache misses
- Memory [access patterns]
<table>
<thead>
<tr>
<th>Access Pattern</th>
<th>Illustrations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block (ND)</td>
<td><img src="image" alt="Block Illustration" /></td>
</tr>
<tr>
<td>Window (ND)</td>
<td><img src="image" alt="Window Illustration" /></td>
</tr>
<tr>
<td>Adjacency</td>
<td><img src="image" alt="Adjacency Illustration" /></td>
</tr>
<tr>
<td>Traversal (BFS, DFS)</td>
<td><img src="image" alt="Traversal Illustration" /></td>
</tr>
<tr>
<td>Permutation</td>
<td><img src="image" alt="Permutation Illustration" /></td>
</tr>
<tr>
<td>Irregular</td>
<td><img src="image" alt="Irregular Illustration" /></td>
</tr>
</tbody>
</table>

- Template class for each access pattern
- Instantiate buffers (e.g., matrix) with appropriate class
- Framework / runtime optimizes data layout

__global__ void Conv2(const float *in, int width, int height, int stride,
    const float *conKernel, float *out) {
    int x = blockIdx.x * BLOCK_WIDTH + threadIdx.x;
    int y = blockIdx.y * BLOCK_HEIGHT + threadIdx.y;
    float result = 0.0f;
    for (int ky = -4; ky <= 4; ++ky) {
        for (int kx = -4; kx <= 4; ++kx) {
            result += in((x + kx - 4) * (y + ky) + (x, ky, 0, width)) *
                conKernel((ky + 4) * 9 + (kx + 4)));
        }
    }
    out[y * stride + x] = result;
}

__shared__ float s_temp[BLOCK_WIDTH + 8] * (BLOCK_HEIGHT + 8);
s_temp[BLOCK_WIDTH + threadIdx.y] = in(CLAMP(y - threadIdx.y - 4, 0, height) * stride -
    if (threadIdx.x < 8) {
        s_temp[BLOCK_WIDTH + threadIdx.y] = in(CLAMP(x - threadIdx.x) * BLOCK_WIDTH -
    }
    if (threadIdx.y < 8) {
        s_temp[BLOCK_WIDTH + threadIdx.y] = in(CLAMP(y - threadIdx.x) *
    }
    if (threadIdx.x > 8) {
        s_temp[BLOCK_WIDTH + threadIdx.y] = in(CLAMP(x + threadIdx.x) * BLOCK_WIDTH -
    }
    __syncthreads();
    float result = 0.0f;
    for (int ky = -4; ky <= 4; ++ky) {
        for (int kx = 0; kx <= 9; ++kx) {
            result += s_temp((BLOCK_WIDTH - threadIdx.x) * (threadIdx.y + kx) +
                threadIdx.y))
        }
    }
    out[y * stride + x] = result;
}

__constant__ float kConvKernel[9];

__global__ void Conv2 MAPS_MULTI(float *maps, int out_width, int out_height, int in_width, int in_height,
    float *kConvKernel, float *in, int stride) {
    int i = 0;
    int oiter = 0.0f;
    MAPS_FOREACH(out, in) {
        oiter += *iter * kConvKernel[i++];
    }
    out.commit(); // Write all outputs to global memory
}

FFMK: Building an Exascale Operating System
Resource prediction:
- CPU cycles [energy?]
- Cache misses
- Memory [access patterns]

Fault tolerance:
- Which data to checkpoint (and when)
- Ability to handle node failures
FFMK prototype runs on real HPC systems

We build **OS/R for Exascale:**
- Low noise
- Load management
- Fault tolerance

Take burden from application developers

ffmk.tudos.org