Practical Experience with the AllScale Programming Model

Thomas Fahringer
Austrian HPC Meeting 2019, Feb. 25 - 27, 2019
AllScale – A FETHPC H2020 Project

- Horizon 2020 funded project and marked “Success Story”
- October 2015 – September 2018
- Coordinated by Thomas Fahringer, DPS Group, Department of Computer Science
- 5 Partners
  - Friedrich-Alexander University Erlangen-Nürnberg
  - IBM Ireland
  - KTH Stockholm
  - Numeca, Brussels
  - Queen’s University Belfast

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No. 671603
Parallel Architectures

Multicore

- OpenMP/Cilk

Accelerators

- OpenCL/CUDA

Clusters

- MPI/PGAS
Real World Architectures

OpenMP/Cilk

OpenCL/CUDA

\[\neq\]

MPI/PGAS
Hybrid Codes

- Issues
  - Hard-coded problem decomposition
  - Lack of coordination among runtime systems
- No built-in support for
  - Performance Portability
  - Auto-tuning
  - Load balancing
  - Monitoring
  - Resilience
  - ...
AllScale Vision

Application → Unified parallel programming model → Toolchain

Parallel algorithm
Portability, tuning, resilience, ...

Memory

www.uibk.ac.at/fz-hpc
Conventional Flat Parallelism

How to map flat parallelism to a hierarchical parallel architecture?
Complex handling of errors – global operations

... global barrier
Recursively Nested Parallelism

Exponential parallel growth

- Global Synchronisation
- Local Synchronisation

... Recursive call
AllScale Platform

Applications

Generic Parallel Primitives (C++ Template API)

API-aware high-level Compiler

Unified Runtime System

Scheduler

Small- to Extreme-Scale Parallel Architectures

Desktop Hardware

Development

Online Monitoring and Analysis

Resilience Management

Tuning & Deployment

Pilot Applications

Single Source User Interface

Generic APIs for abstract Algorithm Descriptions

Code Generation for Accelerators and Distributed Memory

Universal Abstract Machine Model

Dynamic Load, Data and Resource Management

Parallel Hardware

Computation & Data Management

Decomposition & Restructuring

Identify & Express Parallelism

www.uibk.ac.at/fz-hpc
Work and Data

Algorithms manipulate the state of data structures
Parallel Work and Data

Resource A
accessible data
inaccessible data

Resource B

Resource C
Parallel Work and Data

decomposition of workload is main focus of e.g. MPI, OpenMP, Cilk, OpenCL, CUDA,…

data structure decomposition and management left to the user
Parallel Work and Data

Desired: advanced system features and services including
- performance **portability**
- inter-node **load balancing**
- dynamic resource utilization
- resilience to node failures

For those, **runtime systems** require **control** over distribution of **work and data**
DSL/API Design

Operators:
- parallel loops
- parallel reduction
- map-reduce
- stencils
- V-cycles
- …

Data Structures:
- multi-dimensional grids
- adaptive grids
- maps
- trees
- hierarchical meshes
- …

application

parallel constructs
AllScale’s Approach

**Operators:**
- parallel loops
- parallel reduction
- map-reduce
- stencils
- V-cycles
- ...

**Data Structures:**
- multi-dimensional grids
- adaptive grids
- maps
- trees
- hierarchical meshes
- ...

higher order parallel primitive

generalized abstract data structure

primitives for parallel construct
Work Flow Construct

- **prec** – a *higher order function* to
  - express recursive task-based parallelism
  - support fine-grained, hierarchical dependencies
Data Items

User Perspective

- facade

Runtime System Perspective

- logical
- physical
- fragment
- subsets addressed as regions

accessible cell
inaccessible cell

location A
location B
location C
Data Items

User Perspective

 Runtime System Perspective

location A

location B

location C

logical | physical

fragment

subsets addressed as regions

accessible cell

inaccessible cell

www.uibk.ac.at/fz-hpc
Data Items

User Perspective

façade

accessible cell

inaccessible cell

Runtime System Perspective

logical

physical

fragment

location A

subsets addressed as regions

location B

location C

User Perspective Runtime System Perspective
Work and Data Link

recursive work load decomposition defined by algorithm

automatically deduced by the compiler

data dependencies of tasks to be managed by the runtime system
for (int i = 0; i < 10; i++) {
    A[i] += i;
}

Increments the first 10 elements of array A with values 0-9.
A Simple Parallel Loop

```
int rank, size;
MPI_Comm_rank(COM,&rank)
MPI_Comm_size(COM,&size)
int p = 10/size;
MPI_Scatter(A,…)
for (int i = p*rank; i < p*rank+p; i++) {
    A[i] += i;
}
MPI_Gather(A,…)
```

Increments the first 10 elements of array A with values 0-9 in parallel.
Experimental Target Systems

- **Meggie (FAU)**
  - 728 nodes
  - 2x Intel Xeon E5-2630 v4 (10 cores) each
  - 14,560 cores in total
  - Intel OmniPath

- **VSC-3 (UIBK)**
  - 2020 nodes
  - 2x Intel Xeon E5-2650 v2 (8 cores) each
  - 32,320 cores in total
  - QDR-80 dual-link InfiniBand
AMDADOS

- Oil spill simulation, developed with IBM Ireland
- Stencil and kalman filter for assimilating sensor data
- AllScale exceeds performance of MPI reference implementation

Sources: https://www.chemistryworld.com/features/oil-spill-cleanup/3008990.article, Marcel Ritter (UIBK)
- Space weather prediction, developed by KTH Stockholm
- Particle-in-Cell simulation
- Performance improvement compared to MPI reference implementation

Source: https://twitter.com/maven2mars/status/984440044659159040

iPiC3D on Meggie (FAU)

Source: https://twitter.com/maven2mars/status/984440044659159040
AMDADOS on Meggie

8x8 subdomains per node (20 cores each)

throughput per node [subdomains/s]

number of cores

ideal  MPI  ART
iPiC3D on Meggie

32M particles per node (20 cores each)

throughput per node [particles/s]

number of cores

ideal  MPI  ART
AMDADOS on VSC-3

12x12 subdomains per node (16 cores each)

throughput per node [subdomains/s]

number of cores

ideal

MPI

ART

wwwuibkacatfzhpc
iPiC3D on VSC-3

25M particles per node (16 cores each)

throughput per node [particles/s]

number of cores

ideal  MPI  ART

www.uibk.ac.at/fz-hpc
Parallelization Effort

- Significantly reduced amount of parallelism-related application code compared to classical implementation
- Single source base and API for all types of parallelism
Practical Experience: Benefits and Challenges

+ Data management out of the box
+ No explicit communication and data management
+ Automated object serialization
+ Implicit intra- and inter-node load balancing
+ No need to deal with MPI+X
+ Preserve structure of mathematical equations
+ Separation of concerns

- Requires advanced C++ knowledge
- Longer compile times (but once for multiple runs)
- No support for external libraries that employ non-AllScale parallelism
Thanks

- Visit our AllScale website
  http://www.allscale.eu
- Visit our AllScale repositories
  https://github.com/allscale
- Visit our Research Center HPC website
  www.uibk.ac.at/fz-hpc

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No. 671603