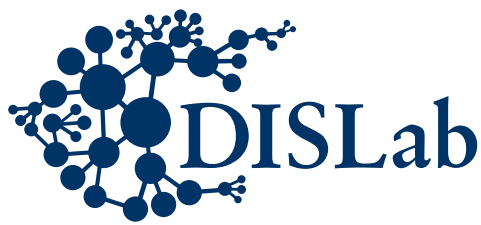
Программная модель Charm++ и ее применимость в эпоху экзаскейла

Александр Фролов

DISLab, АО "НИЦЭВТ"



Школа-семинар "Поиск эффективных суперкомпьютерных архитектур в пост-Муровскую эру", 11 декабря 2017, МИЭМ НИУ ВШЭ

What is “a capable exascale computing system”?

A capable exascale computing system requires an entire computational ecosystem that:

- Delivers 50× the performance of today’s 20 PF systems, supporting applications that deliver high-fidelity solutions in less time and address problems of greater complexity
- Operates in a power envelope of 20–30 MW
- Is sufficiently resilient (average fault rate: ≤1/week)
- Includes a software stack that meets the needs of a broad spectrum of applications and workloads

This ecosystem will be developed using a co-design approach to deliver new software, applications, platforms, and computational science capabilities at heretofore unseen scale

Exascale Challenges[1]

- Power consumption
  - Power is probably the biggest constraint on an exascale system. How can we rein in the power consumed by the individual components as well as the power required for cooling the whole system?
- Resiliency
  - Component failures will be frequent, so that the mean time between failures will be in hours (or minutes!). Can software help applications tolerate these faults?
- Data movement
  - Data movement will become the most expensive operation in coming years in terms of execution time and power costs. How will exascale software hide or mitigate the impact of communication?
- Heterogeneity
  - and dynamic changes to the execution environment will become the norm on future machines. How can software stacks maximize compute resource utilization in the face of such diversity in component capabilities?
- Strong scaling
  - Strong scaling will become the norm at exascale. Problems of interest in many domains don't get larger, but need to be solved faster. In addition, the total memory in a system is expected to grow much slower than compute power. How can we ease the way for applications to transition into this regime?

[1] http://charm.cs.uiuc.edu/why/

Exascale Challenges[2]

- Parallelism
  - Generating parallelism for strong scaling will be a challenge. Exascale compute resources will offer billion-way parallelism. Plain data-decomposition techniques will fall short of providing sufficient parallelism. What fundamental programming paradigm shift(s) are needed to express adequate parallelism?
- Load balancing
  - Load imbalances in applications running on extreme machines will have numerous origins. There will be persistent as well as transient load imbalances. As an example, adaptive refinements to increase simulation fidelity in regions of interest are already a necessity for many applications. How will we maintain high compute throughput in the face of such imbalances?
- Multi-disciplinary applications/seamless modularity
  - Many applications will incorporate multiple modules due to the multi-physics nature of simulations and the need to use existing libraries. Thus there will be a need to cast expensive parallel software into modular, parallel libraries without the seams between the modules translating into performance penalties.

[2] http://charm.cs.uiuc.edu/why/

Why Charm++ for Exascale[3]

A simple idea: **decompose work and data-units into a large number of logical units** and to transfer all work to the powerfull, adaptive **runtime system (RTS)**. This allows to:

- Express more parallelism

[3] http://charm.cs.uiuc.edu/why/

Why Charm++ for Exascale[3]

A simple idea: **decompose work and data-units into a large number of logical units** and to transfer all work to the powerfull, adaptive **runtime system (RTS)**. This allows to:

- Express more parallelism
- Hide latencies

[3] http://charm.cs.uiuc.edu/why/

Why Charm++ for Exascale[3]

A simple idea: **decompose work and data-units into a large number of logical units** and to transfer all work to the powerfull, adaptive **runtime system (RTS)**. This allows to:

- Express more parallelism
- Hide latencies
- Automatically balance load

[3] http://charm.cs.uiuc.edu/why/

Why Charm++ for Exascale[3]

A simple idea: **decompose work and data-units into a large number of logical units** and to transfer all work to the powerfull, adaptive **runtime system (RTS)**. This allows to:

- Express more parallelism
- Hide latencies
- Automatically balance load
- Automatically handles heterogeneous systems

[3] http://charm.cs.uiuc.edu/why/

Why Charm++ for Exascale[3]

A simple idea: **decompose work and data-units into a large number of logical units** and to transfer all work to the powerfull, adaptive **runtime system (RTS)**. This allows to:

- Express more parallelism
- Hide latencies
- Automatically balance load
- Automatically handles heterogeneous systems
- Provide power controll on application level

[3] http://charm.cs.uiuc.edu/why/

Why Charm++ for Exascale[3]

A simple idea: **decompose work and data-units into a large number of logical units** and to transfer all work to the powerfull, adaptive **runtime system (RTS)**. This allows to:

- Express more parallelism
- Hide latencies
- Automatically balance load
- Automatically handles heterogeneous systems
- Provide power controll on application level
- Provide resiliency

[3] http://charm.cs.uiuc.edu/why/

Why Charm++ for Exascale[3]

A simple idea: **decompose work and data-units into a large number of logical units** and to transfer all work to the powerfull, adaptive **runtime system (RTS)**. This allows to:

- Express more parallelism
- Hide latencies
- Automatically balance load
- Automatically handles heterogeneous systems
- Provide power controll on application level
- Provide resiliency
- Achieve modularity with high performance

[3] http://charm.cs.uiuc.edu/why/

Why Charm++ for Exascale[3]

A simple idea: **decompose work and data-units into a large number of logical units** and to transfer all work to the powerfull, adaptive **runtime system (RTS)**. This allows to:

- Express more parallelism
- Hide latencies
- Automatically balance load
- Automatically handles heterogeneous systems
- Provide power controll on application level
- Provide resiliency
- Achieve modularity with high performance
- Enhance productivity

[3] http://charm.cs.uiuc.edu/why/

Язык параллельного программирования Charm++

- История
  - Parallel Programming Laboratory at the University of Illinois
  - создание – начало 90-х годов, open-source
  - текущая версия 6.8.2
  - Charmworks Inc.

Язык параллельного программирования Charm++

- История
  - Parallel Programming Laboratory at the University of Illinois
  - создание – начало 90-х годов, open-source
  - текущая версия 6.8.2
  - Charmworks Inc.
- Основные принципы Charm++
  - объектная ориентированность (расширяет C++)
  - управление потоком асинхронных сообщений
  - ориентированность на мелкозернистый параллелизм (overdecomposition)

Язык параллельного программирования Charm++

- История
  - Parallel Programming Laboratory at the University of Illinois
  - создание – начало 90-х годов, open-source
  - текущая версия 6.8.2
  - Charmworks Inc.
- Основные принципы Charm++
  - объектная ориентированность (расширяет C++)
  - управление потоком асинхронных сообщений
  - ориентированность на мелкозернистый параллелизм (overdecomposition)
- Специфические возможности
  - динамическая балансировка нагрузки
  - поддержка отказоустойчивости (сохранение контрольных точек)

Язык параллельного программирования Charm++

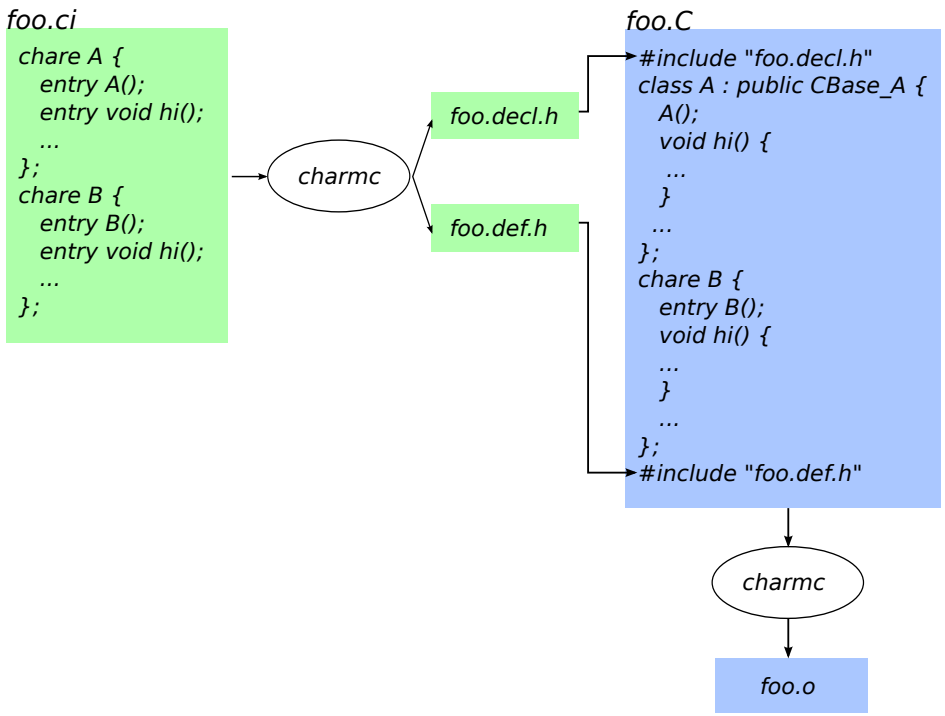
- История
  - Parallel Programming Laboratory at the University of Illinois
  - создание – начало 90-х годов, open-source
  - текущая версия 6.8.2
  - Charmworks Inc.
- Основные принципы Charm++
  - объектная ориентированность (расширяет C++)
  - управление потоком асинхронных сообщений
  - ориентированность на мелкозернистый параллелизм (overdecomposition)
- Специфические возможности
  - динамическая балансировка нагрузки
  - поддержка отказоустойчивости (сохранение контрольных точек)
- Поддерживаемые типы HPC-систем
  - SMP-узлы с NUMA памятью
  - кластеры с Infiniband, BlueGene/P, BlueGene/Q, Cray XK, Cray XC
  - ведутся работы по поддержке ускорителей (Xeon Phi, GPU)

Язык параллельного программирования Charm++

- История
  - Parallel Programming Laboratory at the University of Illinois
  - создание – начало 90-х годов, open-source
  - текущая версия 6.8.2
  - Charmworks Inc.
- Основные принципы Charm++
  - объектная ориентированность (расширяет C++)
  - управление потоком асинхронных сообщений
  - ориентированность на мелкозернистый параллелизм (overdecomposition)
- Специфические возможности
  - динамическая балансировка нагрузки
  - поддержка отказоустойчивости (сохранение контрольных точек)
- Поддерживаемые типы HPC-систем
  - SMP-узлы с NUMA памятью
  - кластеры с Infiniband, BlueGene/P, BlueGene/Q, Cray XK, Cray XC
  - ведутся работы по поддержке ускорителей (Xeon Phi, GPU)
- Приложения
  - NAMD, OpenAtom, ChANga, EpiSimdemics
  - BigSim, ClothSim, имитационная модель сети "Ангара"

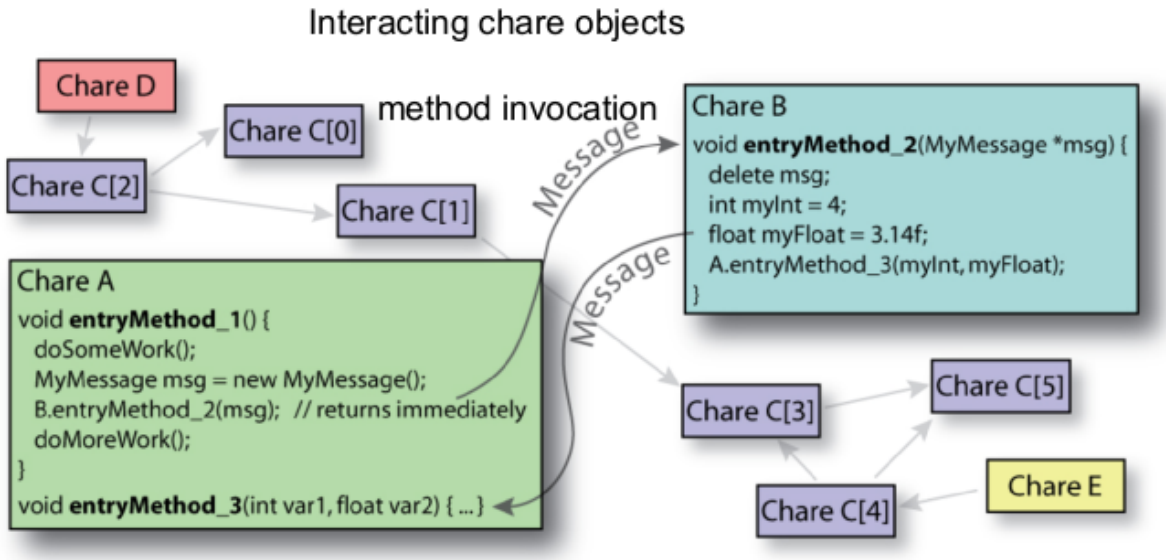
Программная модель Charm++ (1)

- Базовый объект Charm++ – *chare*
  - приложение состоит из множества *chare*-объектов
  - имеет набор *entry* методов, определенных .ci файле
  - *chare*-объекты обмениваются сообщениями, вызывая *entry*-методы друг друга
  - *entry*-методу доступны на запись только данные, принадлежащие соответствующему *chare*-объекту
  - выполнение *entry*-методов не может быть прервано, что гарантирует атомарность изменения данных внутри *chare*-объекта
  - могут быть объединены в "коллекции": 1D/2D/../7D-массивы

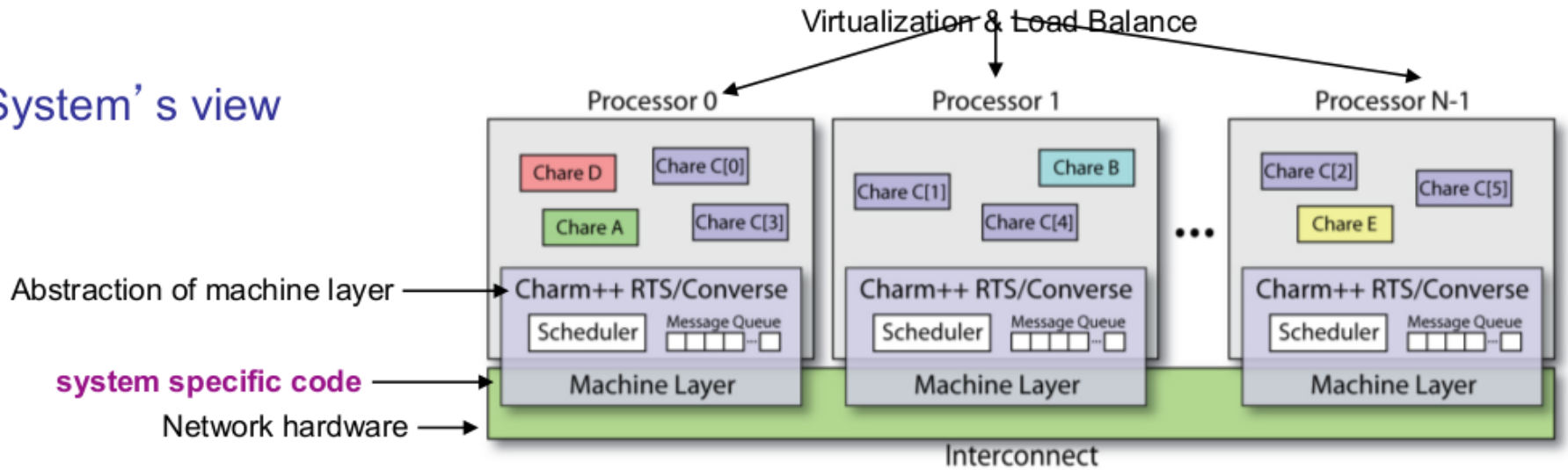


Программная модель Charm++

Application's view



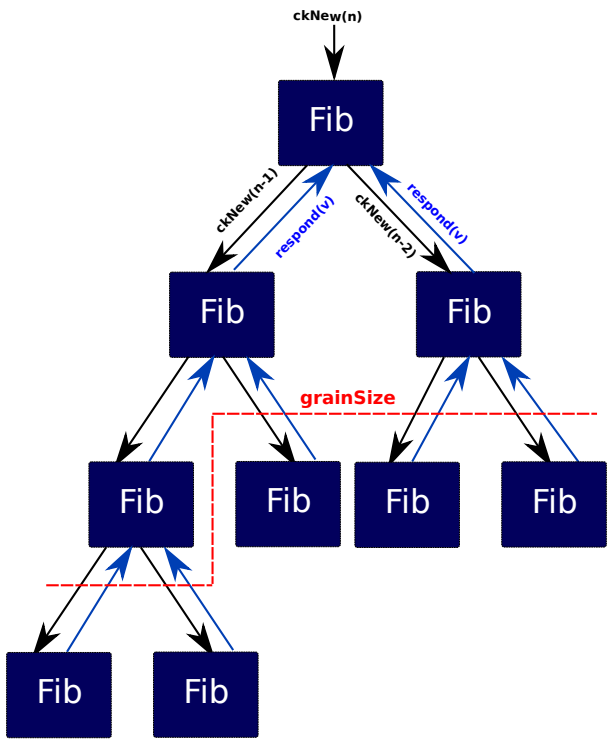
System's view



Charm++ Example

Fibonacci (fib.ci)

```
1   readonly int grainSize;
2   mainmodule fib {
3     mainchare Main {
4       entry Main(CkArgMsg* m);
5     };
6     chare Fib {
7       entry Fib(int n, bool isRoot, CProxy_Fib parent);
8       entry void respond(int value);
9     };
10  };
```



Charm++ Example

Fibonacci (fib.C)

```
1  int grainSize;
2  struct Main : public CBase_Main {
3    Main(CkArgMsg* m) {
4      grainSize = 10;
5      CProxy_Fib::ckNew(atoi(m->argv[1]), true, CProxy_Fib());
6    }
7  };
8  struct Fib : public CBase_Fib {
9    CProxy_Fib parent; bool isRoot;
10   Fib(int n, bool isRoot_, CProxy_Fib parent_)
11     : parent(parent_), isRoot(isRoot_), result(0), count(2) {
12     if (n < grainSize) respond(FibSeq(n));
13     else {
14       CProxy_Fib::ckNew(n - 1, false, thisProxy);
15       CProxy_Fib::ckNew(n - 2, false, thisProxy);
16     }
17   }
18   void respond(int val) {
19     result += val;
20     if (--count == 0 || n < grainSize) {
21       if (!isRoot) {
22         parent.response(val);
23         delete this;
24       } else {
25         CkPrintf("Fibonacci number is: %d\n", val);
26         CkExit();
27       }
28     }
29   }
30 };
```

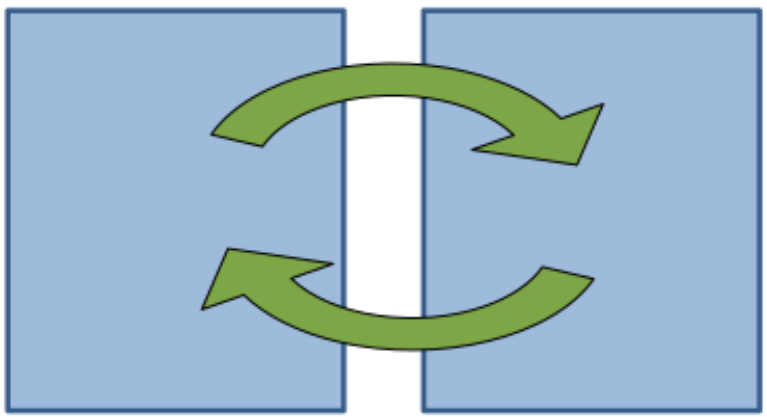
Charm++ Example

2D Jacobi (2D decomposition)

Use two interchangeable matrices

```
do {
   computeKernel();
   maxDiff = max(abs (A - B));
} while (maxDiff > DELTA);

computeKernel() {
   foreach i,j {
      B[i,j] = (A[i,j] +
                A[i+1,j] +
                A[i-1,j] +
                A[i,j+1] +
                A[i,j-1]) / 5;
   }
   swap (A, B);
}
```



Charm++ Example

2D Jacobi (2D decomposition)

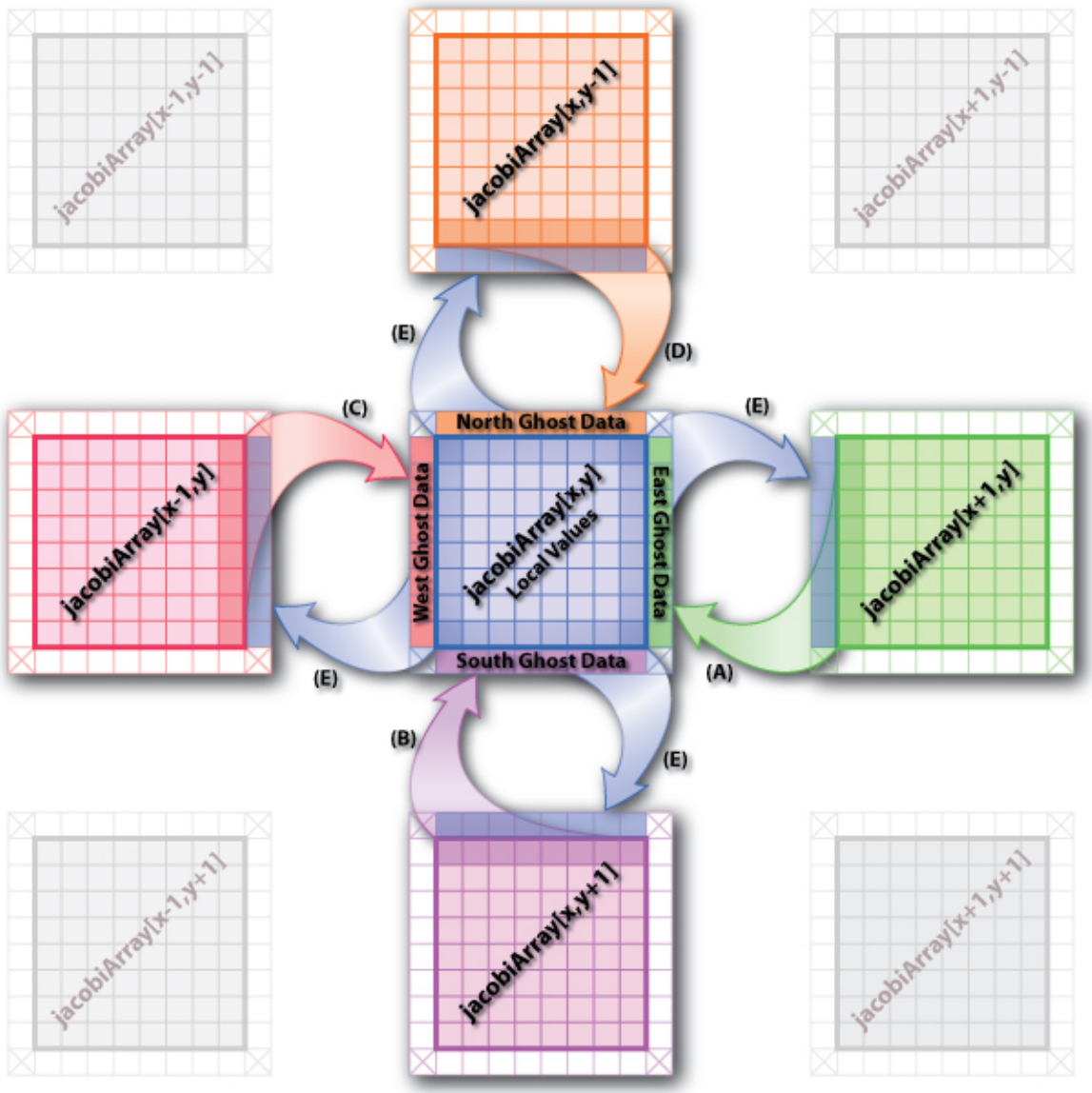
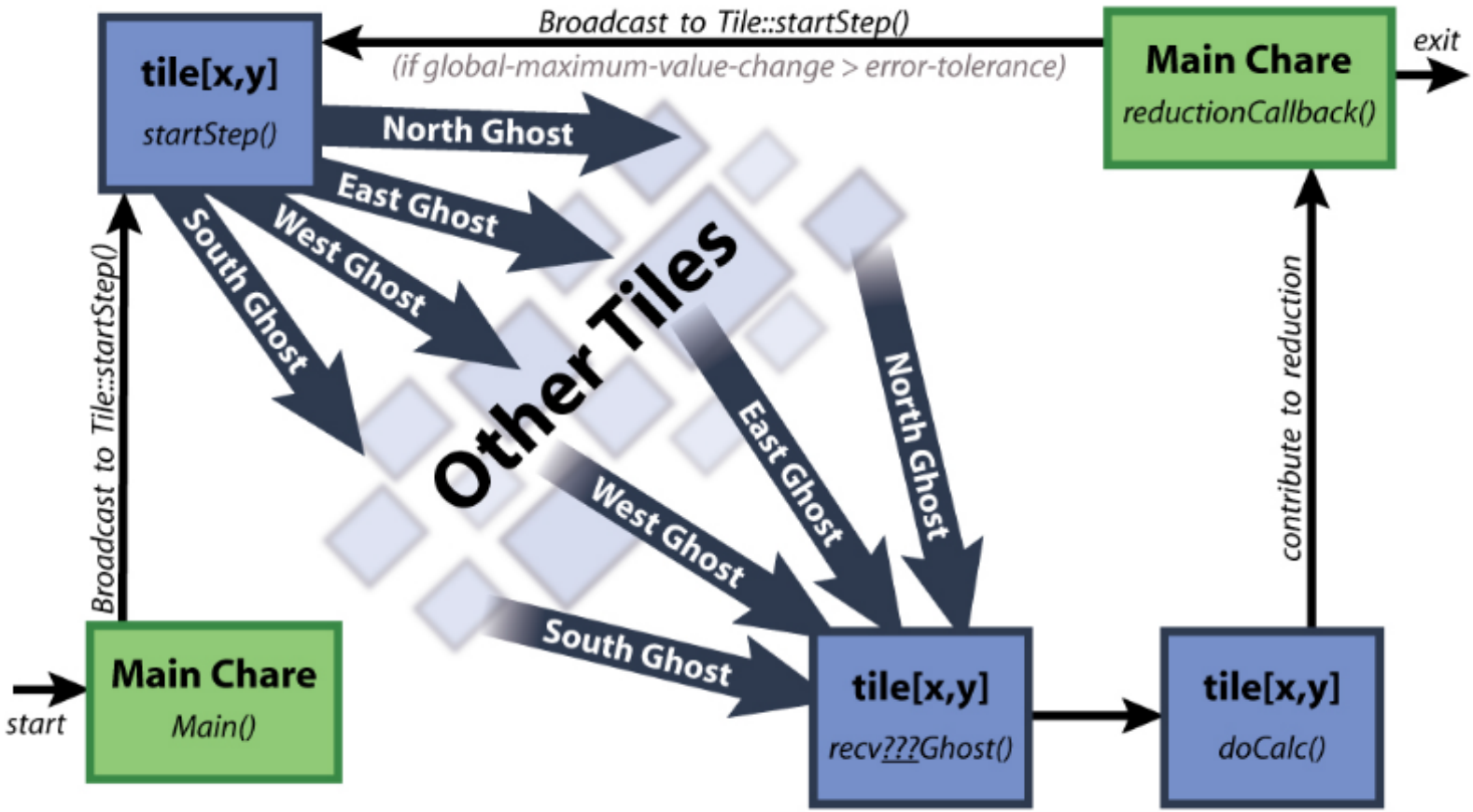


Figure 3: Communication Pattern (Single Chare, Single Step)

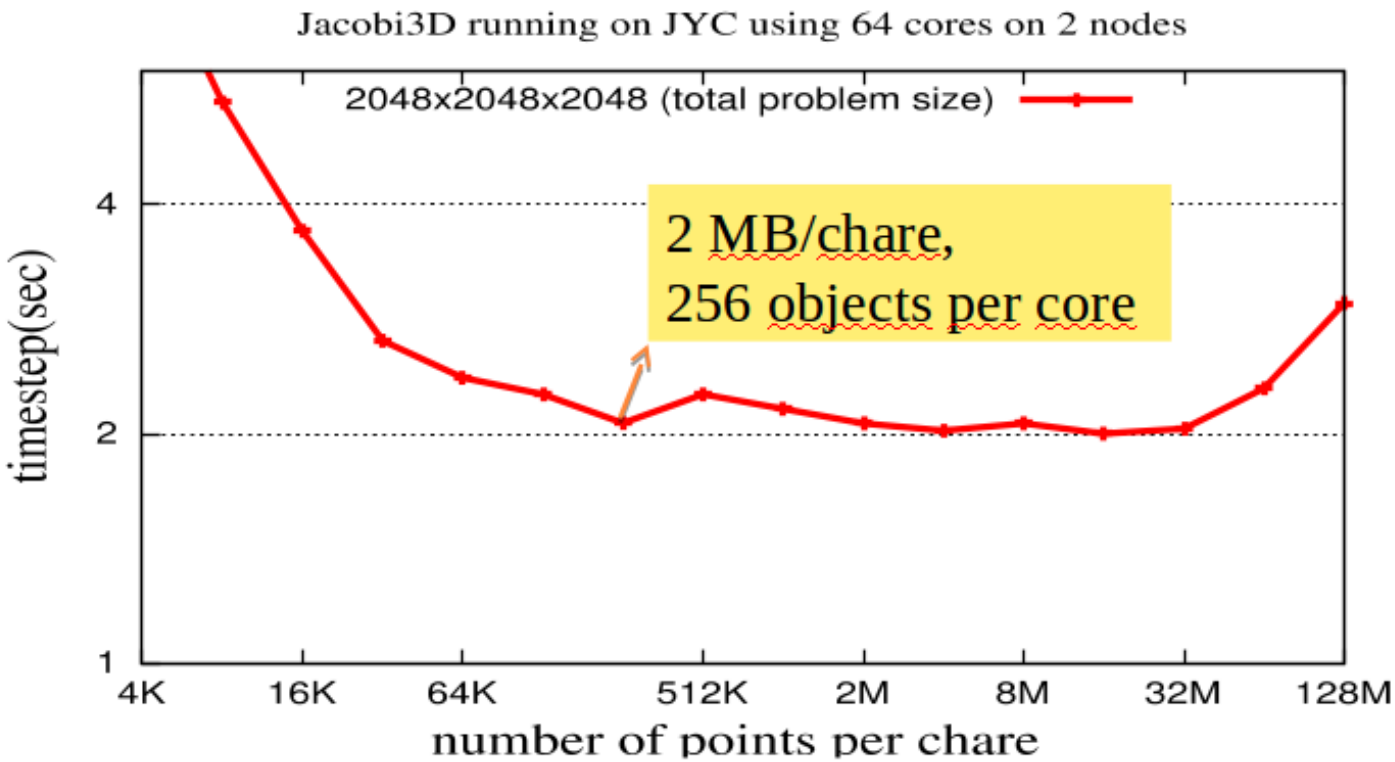
Charm++ Example

2D Jacobi



Charm++ Example

2D Jacobi



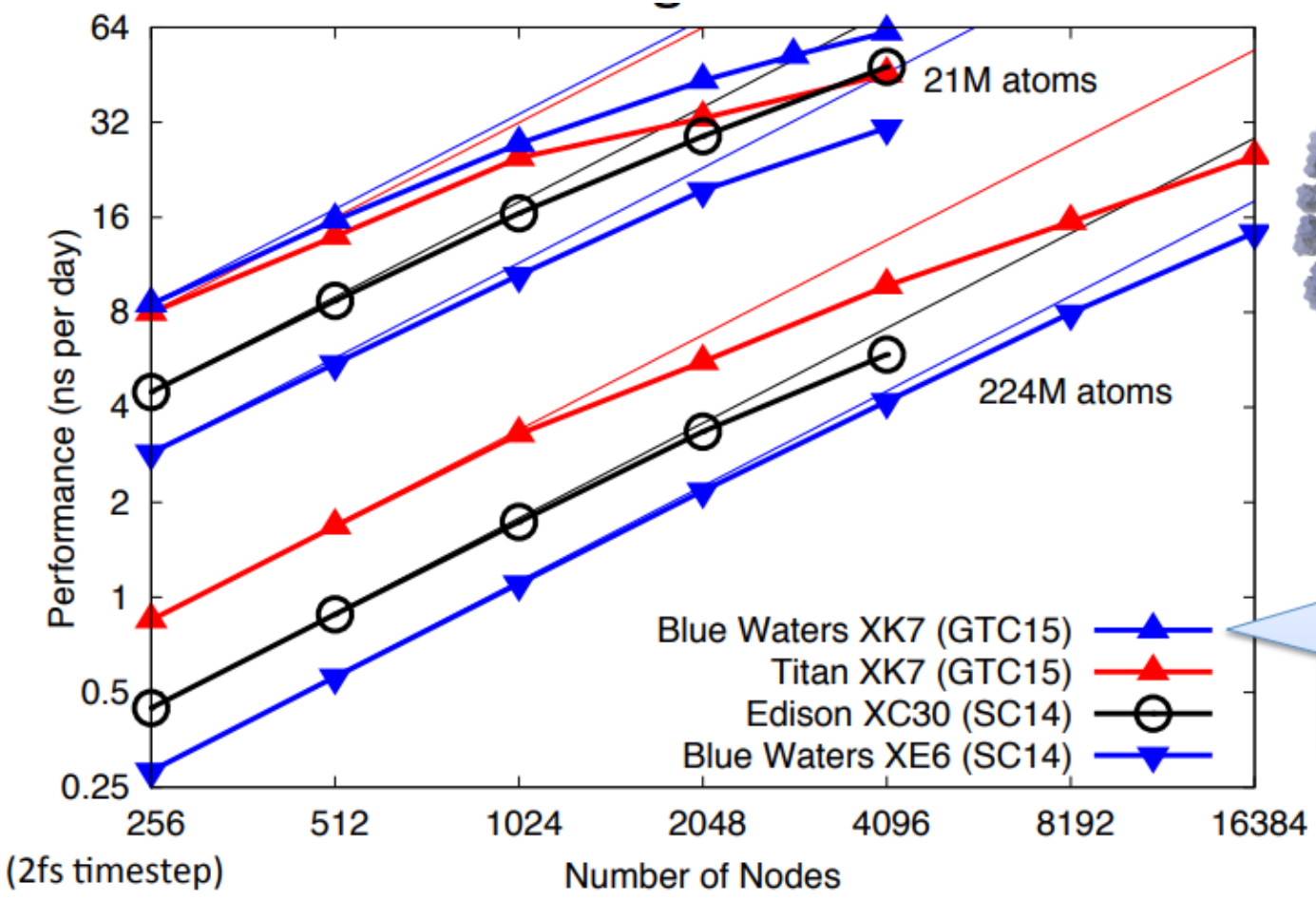
Dynamic Load Balancing

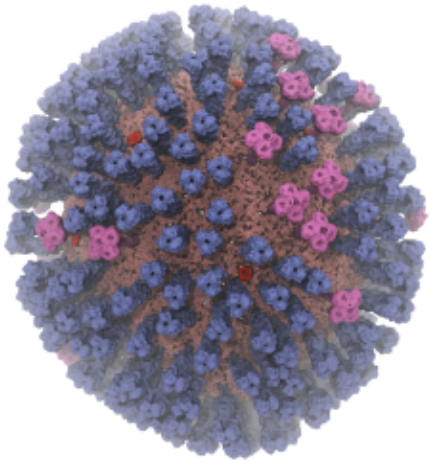
- LBDatabase – load balancer database (chare group) stores LB instrumented data;
- LB strategies (inherits from BaseLB):
  - centralized:
    - RandCentLB, MetisLB, ScotchLB, GreedyLB, GreedyRefineLB, GreedyCommLB, TopoCentLB, RefineLB, RefineSwapLB, RefineCommLB, RefineTopoLB, BlockLB, RotateLB, ComboCentLB,...
  - distributed:
    - NeighborLB, WSLB, DistributedLB
  - hierarchical:
    - HybridLB
- MetaLB
  - Metabalancer (uses a linear prediction model to set the load balancing period) can choose which strategy to use (GreedyLB, RefineLB, HybridLB, DistributedLB, MetisLB and ScotchLB).
- Invocation methods:
  - Periodical mode: called by runtime periodically, controlled by `LBPeriod`
  - As sync mode: when all chare array elements call `AtSync()` load balancing is triggered, `ResumeFromSync()` is called in each element after balancing is completed.
  - Manual mode: `CkStartLB()` starts load balancing immediately.

Charm++ in Production

Application	Domain	Previous parallelization
NAMD	Classical MD	PVM
ChaNGa	N-body gravity & SPH	MPI
EpiSimdemics	Agent-based epidemiology	MPI
OpenAtom	Electronic Structure	MPI
Spectre	Relativistic MHD	
FreeON/SpAMM	Quantum Chemistry	OpenMP
Enzo-P/Cello	Astrophysics/Cosmology	MPI
ROSS	PDES	MPI
SDG	Elastodynamic fracture	
ADHydro	Systems Hydrology	
Disney ClothSim	Textile & rigid body dynamics	TBB
Particle Tracking	Velocimetry reconstruction	
JetAlloc	Stochastic MIP optimization	

NAMD Scalability



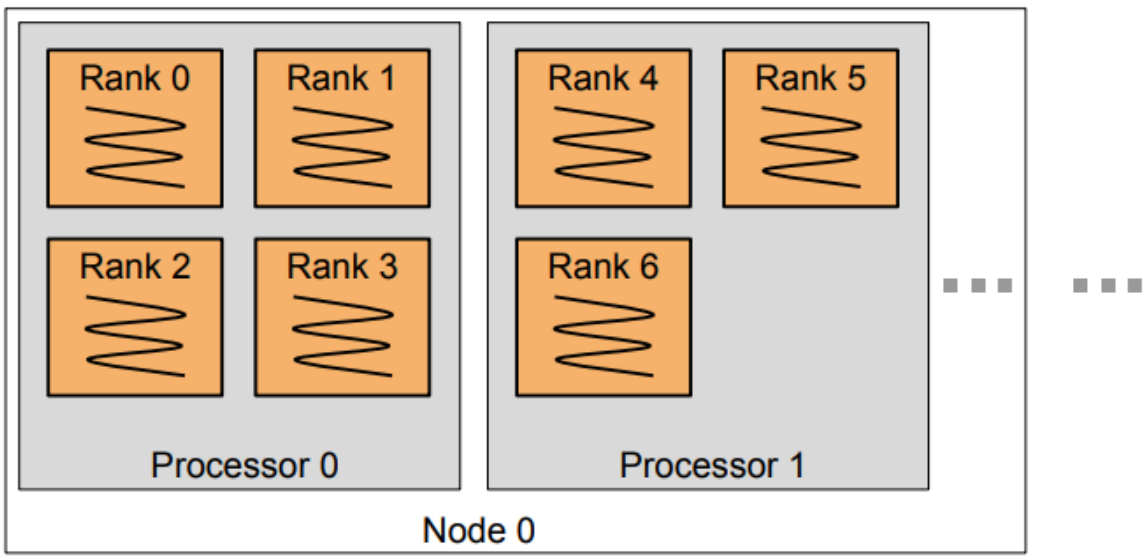


Influenza, 210M atoms
Amaro Lab, UCSD

Topology-aware scheduler

Adaptive MPI[4]

- MPI implementation on top of Charm++
  - AMPI virtualizes MPI ranks, allowing multiple ranks to execute per core
  - AMPI virtualizes ranks as threads => Thread Safety: AMPI programs are MPI programs without mutable global/static variables => refactoring of legacy MPI codes needed
  - Message-driven execution (`MPI_Send`)
  - Dynamic load-balancing, checkpointing
- Applications: LLNL proxy apps, Harm3D (black hole simulations), PlasmaComCM (plasma-coupled combustion simulation)



[4] Source: S.White, Adaptive MPI Performance & Application Studies, Charm++ Workshop 2017

Adaptive MPI

- Conformance:
  - AMPI supports the MPI-2.2 standard
  - MPI-3.1 nonblocking & nbor collectives
  - User-defined, non-commutative reductions ops
  - Improved derived datatype support
- Performance:
  - More efficient (all)reduce & (all)gather(v)
  - More communication overlap in `MPI_{Wait,Test}{any,some,all}` routines
  - Point-to-point messaging, via Charm++'s new zero-copy RDMA send API

Библиотека Topological Routing and Aggregation Module (TRAM)

- TRAM – библиотека для Charm++ приложений (NDMeshStreamer)
- Разработана для улучшения результата HPC Challenge (2011)
- Возможности TRAM:
  - агрегация коротких сообщений
  - создание виртуальных топологий (торовых) на которую отбражаются chare-объекты
  - узлы виртуальной топологии отображаются на узлы кластера, аллоцированные для приложения
  - агрегированные сообщения передаются peer-to-peer
  - для передачи данных с использованием TRAM используется специальный API (не стандартные вызовы entry-методов)
- Поддержка выделения методов ключевым словом `aggregate` для использования TRAM, методы должны иметь единственный параметр фиксированного размера (с 6.8.0)

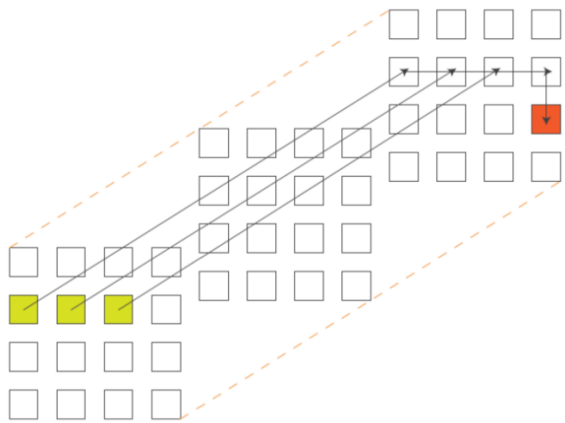


Рис.: Маршрутизация сообщений в TRAM (3D-тор)

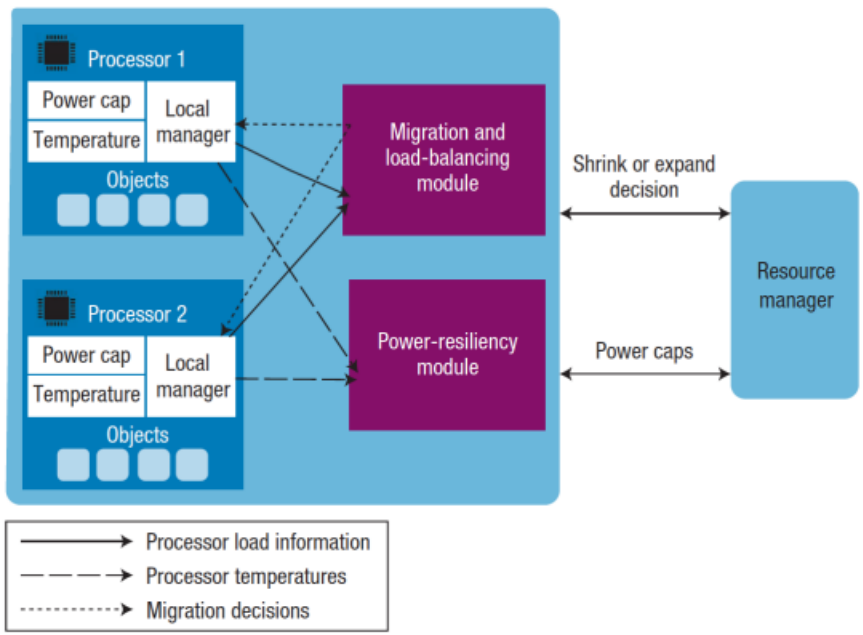
GPU Support in Charm++ [5]

- GPU Manager
  - GPU task Management library
  - Register kernel for asynchronous invocation
  - Automates data movement
  - Overlap kernel execution and data transfer
  - Pre-allocated pool of pinned memory
  - Runtime profiling integration (Projections)
- Accel framework
  - Generate code from tagged entry methods (Host (CPU) and device (CUDA), Extend with tuning keywords, Annotate object data access)
  - Builds on GPU manager
  - Batch fine grained kernel launches

[5] M.Robson, Heterogeneous Computing in Charm++, Charm++ Workshop 2017

Power-awareness of Charm++ Runtime System [6]

- Charm++ has three main components:
  - **Local manager:** tracks local information such as object loads, CPU temperatures
  - **Load-balancing module:** makes load-balancing decisions and redistributes load
  - **Power-resiliency module:** ensures that the CPU temperatures remain below the temperature threshold, change the power cap

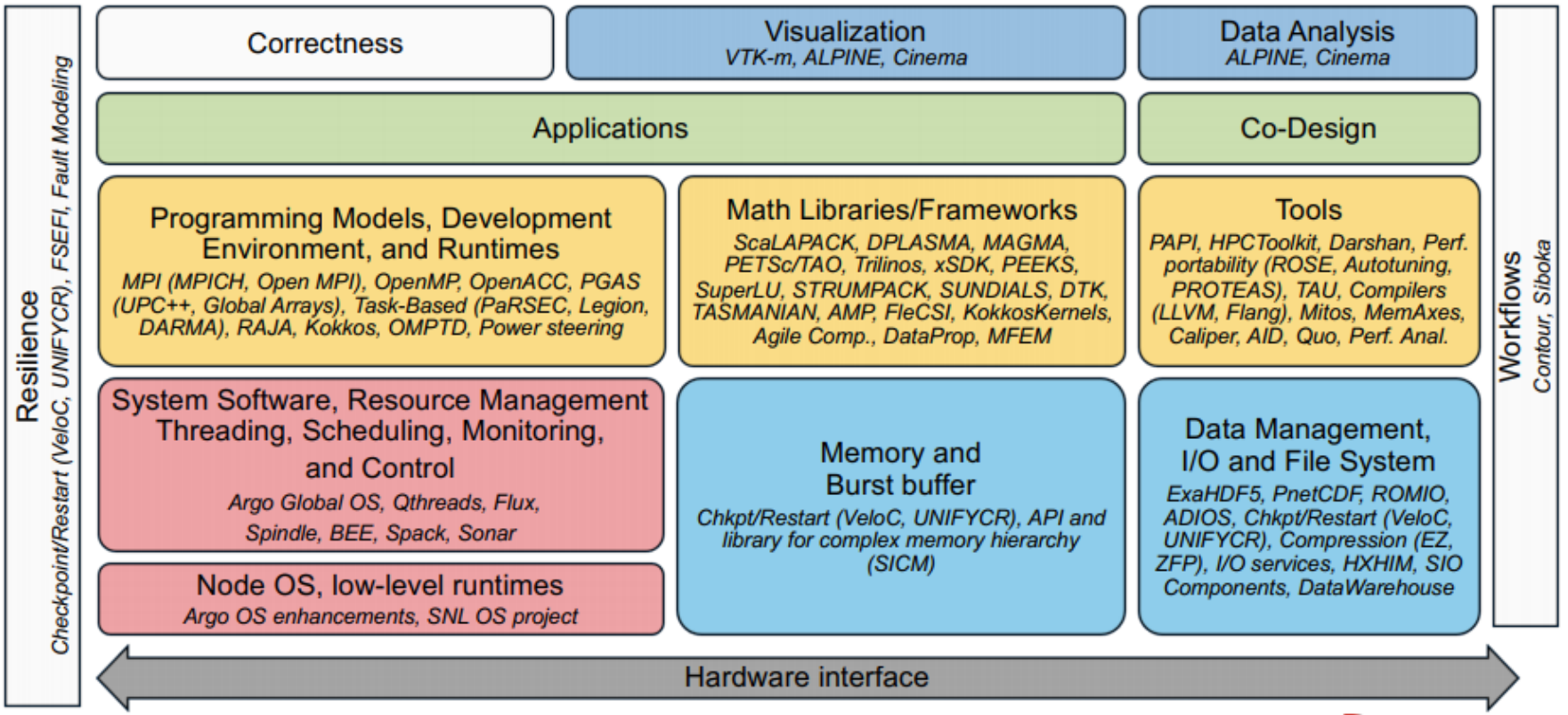


[6] B. Acun and A. Langer and E. Meneses and H. Menon and O. Sarood and E. Totoni and L. V. Kale Power, Reliability, and Performance: One System to Rule them All. Computer. 2016, doi.ieeecomputersociety.org/10.1109/MC.2016.310

Other Charm++ Initiatives

- Porting High-level Languages to Charm++ (Charj, Green-Marl (graph DSL), ...)
- Porting Charm++ on Argobots (user-level thread library from ANL)
- OpenMP intergration into Charm++
- Multi-phase quiescence detection mechanism
- Out-of-core application support
- Support for new levels of memory hierarchy (HBM)
- Applications, applications, and applications...

DOE Exascale Computing Project (ECP) Software Stack



Выводы

- Модель Charm++ обладает рядом очень интересных свойств (асинхронность, управление потоком сообщений, избыточный параллелизм, миграция объектов), перспективных для экзамасштабных вычислений.
  - В Charm++ за счет повышения уровня абстракции позволяет использовать runtime-систему для умного планирования потоков вычислений, обеспечения балансировки вычислений и отказоустойчивости.
  - В Charm++ осуществляеся “передача вычислений” (а не данных), что позволяет экономить ресурс пропускной способности интерконнекта и обеспечить локализацию вычислений.
- В теории Charm++ отвечает почти всем вызовам “экзаскейла”, но остается в большей степени академической разработкой (возможно, это есть самая правильная ниша Charm++).
- Charm++ значительно уступает по размеру сообщества, кодовой базе прикладного программного обеспечения, инструментальным средствам более традиционным (стандартизованным) программным моделям (MPI, OpenMP) и, врядли, это изменится.