Charm++

Migratable Objects + Asynchronous Methods + Adaptive Runtime = Performance + Productivity

Laxmikant V. Kale*

Anshu Arya Nikhil Jain Akhil Langer Jonathan Lifflander

Harshitha Menon Phil Miller Xiang Ni Yanhua Sun

Ehsan Totoni Ramprasad Venkataraman* Lukasz Wesolowski

Parallel Programming Laboratory

Department of Computer Science

University of Illinois at Urbana-Champaign

*{kale, ramv}@illinois.edu

Metrics: Performance and Productivity

Our Implementations in Charm++

	Productivity					Performance		
Code	C++	CI	Benchmark Subtotal	Driver	Total	Machine	Max Cores	Performance Highlight
Required Benchmarks								
1D FFT	54	29	83	102	185	BG/P BG/Q	64K 16K	2.71 TFlop/s 2.31 TFlop/s
Random Access	76	15	91	47	138	BG/P BG/Q	128K 16K	43.10 GUPS 15.00 GUPS
Dense LU	1001	316	1317	453	1770	XT5	8K	55.1 TFlop/s (65.7% peak)
Additional Benchmarks								
Molecular Dynamics	571	122	693	n/a	693	BG/P BG/Q	128K 16K	24 ms/step (2.8M atoms) 44 ms/step (2.8M atoms)
AMR	1126	118	1244	n/a	1244	BG/Q	32k	22 steps/sec, 2d mesh, max 15 levels refinement
Triangular Solver	642	50	692	56	748	BG/P	512	48x speedup on 64 cores with helm2d03 matrix

C++ Regular C++ code
CI Parallel interface descriptions and control flow DAG

Metrics++: Differentiating Capabilities

Demonstrated Productivity Benefits

This year's submission demonstrates:

- Interoperating with MPI
- Asynchronous, non-blocking collectives
- Dynamic load balancing with closed loop control system
- Automatic checkpoints for split execution
- Tolerating process failures

Metrics++: Differentiating Capabilities

Demonstrated Productivity Benefits

This year's submission demonstrates:

FFT	Interoperating with MPI
FFT, RA, LU	Asynchronous, non-blocking collectives
MD, AMR	Dynamic load balancing with closed loop control system
MD	Automatic checkpoints for split execution
MD, AMR	Tolerating process failures

This year's submission demonstrates:

FFT Interoperating with MPI

- gradual, incremental adoption path

FFT, RA, LU Asynchronous, non-blocking collectives

- trivially overlaps collectives with computation

MD, AMR Dynamic load balancing with closed loop control system

- automates "when" and "how" decisions

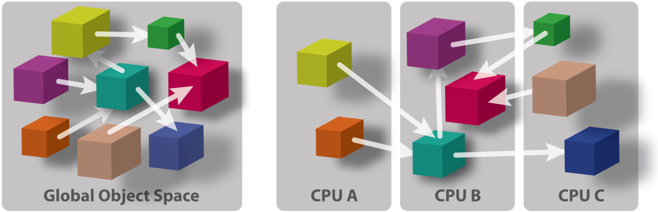
MD Automatic checkpoints for split execution

- checkpoint on x cores, automatic restart on y cores

MD, AMR Tolerating process failures

- continues execution despite randomly injected failures

Charm++ Programming Model



Migratable Objects

- Globally visible
- Migratable
- Overdecomposed
- Data / Task

Async Methods

- Invoke remotely
- Non-blocking
- Non-preemptive
- Prioritized

Adaptive Runtime

- Message-driven
- Orchestrates
- Observes
- Adapts

Ensuing Benefits

overlap, load balance, adaptivity, fault tolerance, elegance, modularity, composability ...

Required Benchmarks

- FFT
- Random Access
- Dense LU Factorization

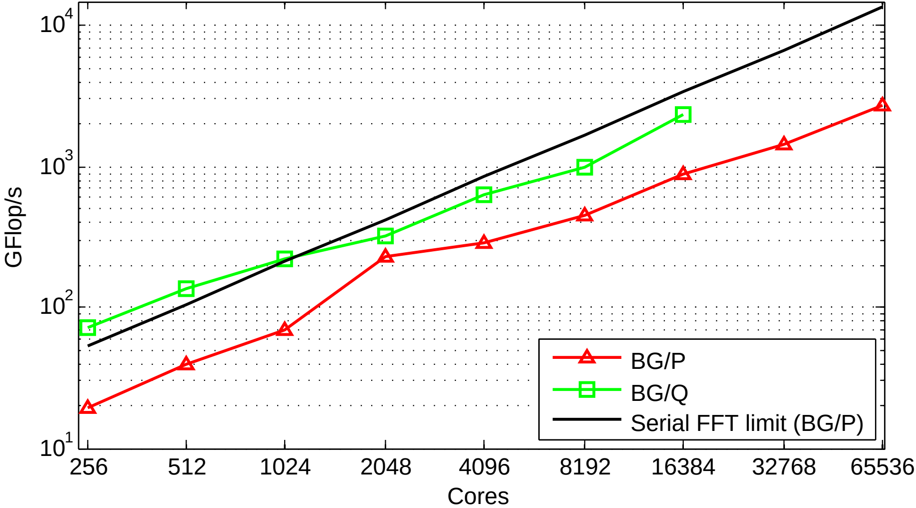
1D FFT (sloc: 83)

dParallel Coordination Code

```
for(phase = 0; phase < 3; ++phase) {
   serial { initStreamer(streamer); }
   when streamerReady() serial {
       sendTranspose(...);
   }
   for(count = 0; count < P; ++count)
       when recvData[phase] (...)  serial {
           applyTranspose(data, n, src);
       }
   if (phase < 2) serial {
       fftw_execute(plan);
       if(phase == 0)
           twiddle();
   }
}
```

1D FFT

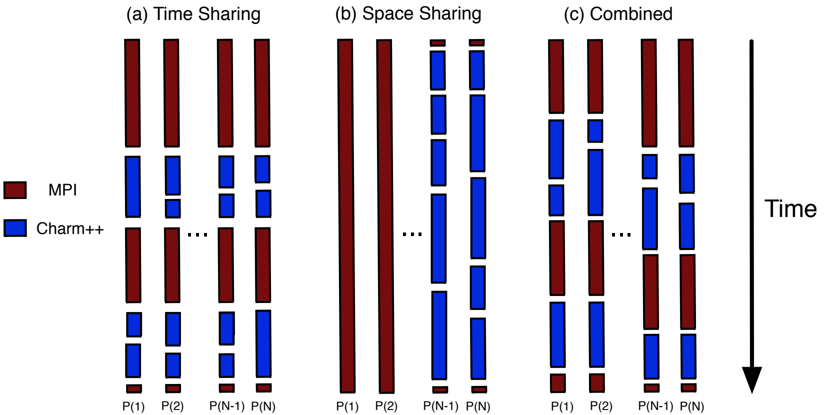
IBM BG/P (Intrepid), BG/Q (Vesta), 25% memory, ESSL /w fftw wrappers



Demonstrated Capability: Interoperability

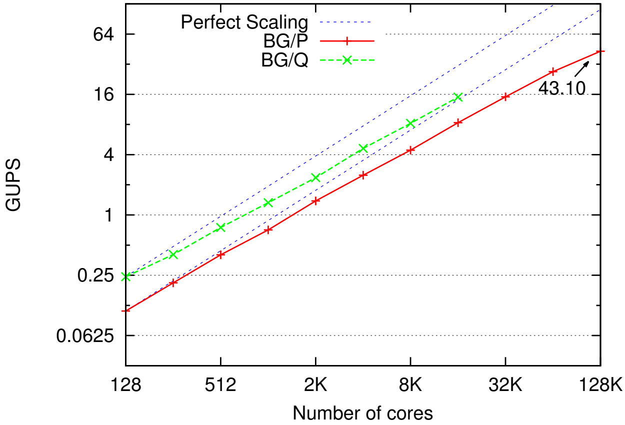
Invoke Charm++ modules from MPI

- Callable like any external MPI library
- Incremental adoption
- Mix-and-match models to needs



Random Access (sloc: 91)

IBM BG/P (Intrepid), BG/Q (Vesta)

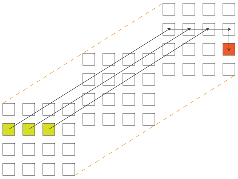


Expressed in < 100 SLOC as point-to-point sends, with optimizations left to the Charm++ TRAM library

Demonstrated Capability: Non-blocking Collectives

Asynchronous, Non-blocking, Topology-aware, Combining, Streaming Many-to-many

- Use point to point sends and let Charm++ optimize communication
- Automatically detect and adapt to network topology of partition
- Topological Routing and Aggregation Module (TRAM)
  - Aggregation of fine-grained communication (Random Access)
  - Minimal topology-aware software routing with recombining at intermediate destinations
  - Streaming to lower memory use and improve network bandwidth utilization
  - Performance improved up to 10x for Random Access and FFT



Dense LU (sloc: 1317)

Complete parallel control flow expressed in 306 lines!

Dense LU (sloc: 1317)

Complete parallel control flow expressed in 306 lines!

- Block-centric
  - Algorithm from a block's perspective
  - Agnostic of processor-level considerations
- Memory-constrained adaptive lookahead
- Flexible data placement

Composable library

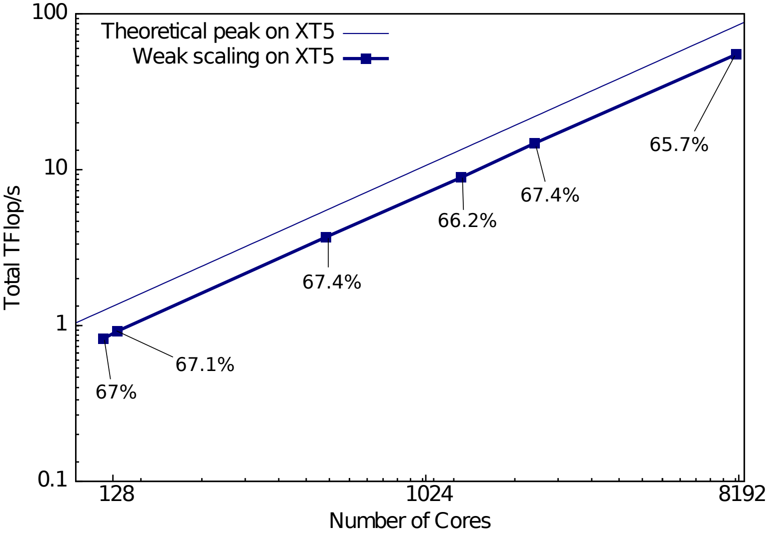
- Modular program structure
- Seamless execution structure (interleaved modules)

Separation of concerns

- Domain specialist codes algorithm
- Systems specialist codes tuning, resource mgmt etc

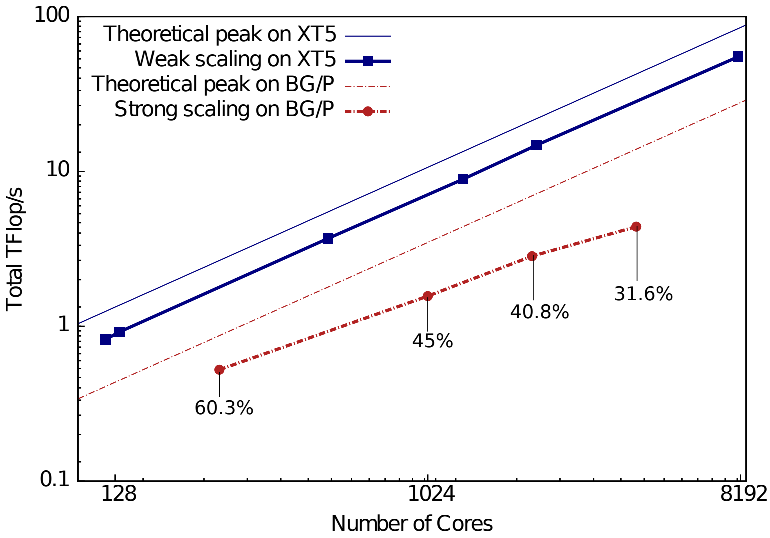
Dense LU

Weak Scaling: (N such that matrix fills 75% memory)



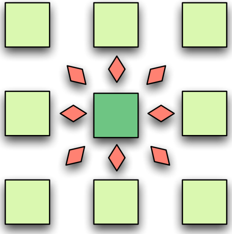
Dense LU

... and strong scaling too! (N=96,000)



LeanMD (sloc: 693)

1. Mimics short-range force calculation in NAMD
2. Similar to Mantevo's *miniMD* (SLOC≈3000), but in < 700 SLOC



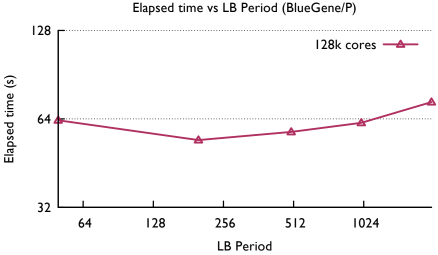
Hybrid force-spatial decomposition

Demonstrated Capability: Automatic, Load Balancing

How to...

Call `AtSync()` in code every iteration

Pass `+MetaLB` on command line

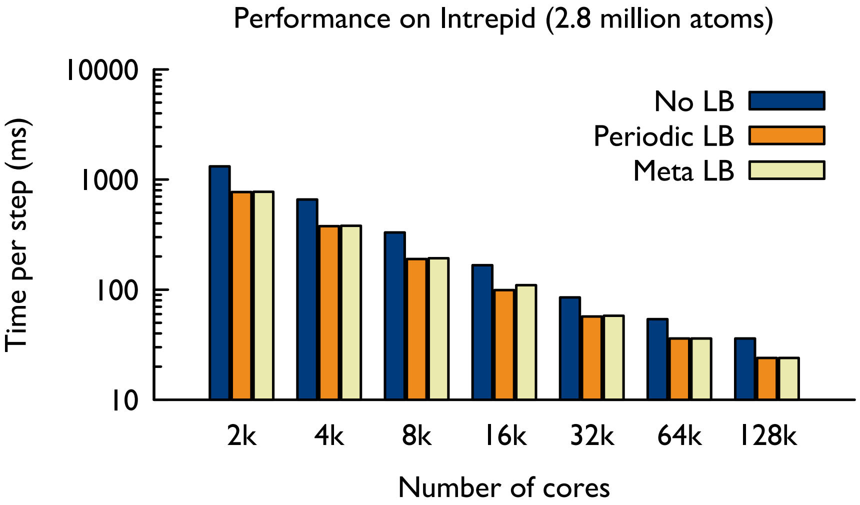


Cores	Periodic (s)	MetaLB (s)
8k	504	413
16k	260	277
32k	131	131
64k	104	100
128k	54	52

- Too frequent load balancing increases total execution time
- Infrequent load balancing leads to load imbalance
- Meta-Balancer adaptively performs load balancing to obtain best total execution time

LeanMD

2.8 million atoms. IBM BG/P (Intrepid)

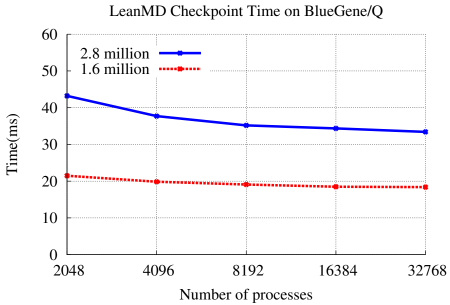


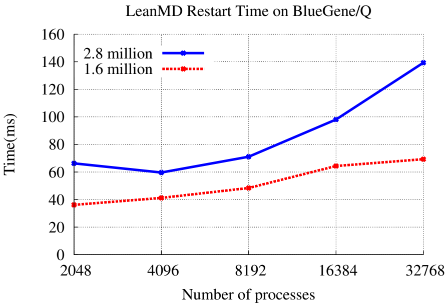
Demonstrated Capability: Tolerating a Failed Process

Checkpoint and Restart Time

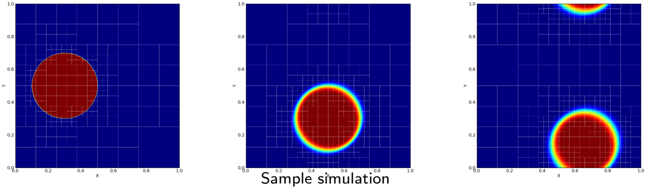
How to...

Call `CkStartMemCheckpoint()` at desired checkpoint interval

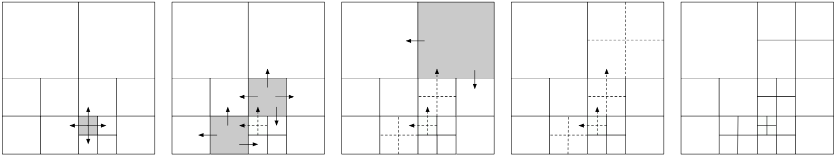




Adaptive Mesh Refinement (sloc: 1244)



Sample simulation



Propagation of refinement decision messages

Adaptive Mesh Refinement

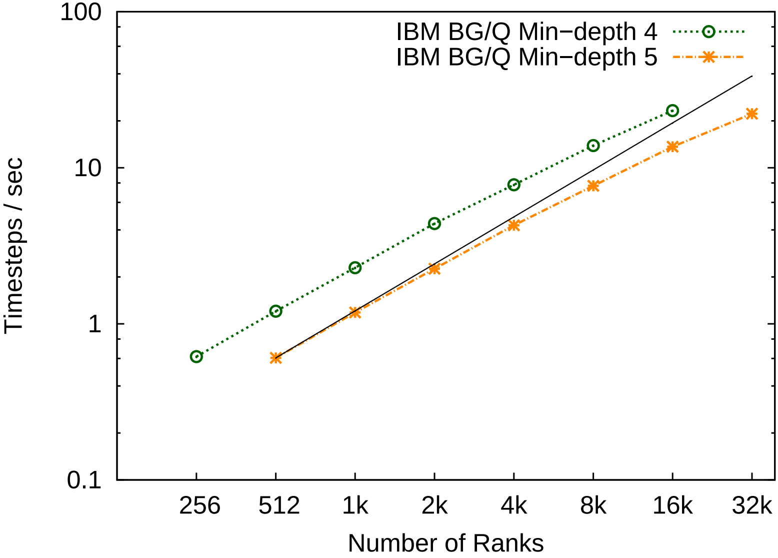
- Block-centric
  - Algorithm from a block's perspective
  - Agnostic of processor-level considerations
- Easily address a mesh-block with bit-vector indices
  - Charm++ handles physical locations
- Dynamic, distributed load balancing

Algorithmic Improvements[1]

- $O(\#blocks/P)$ **vs** $O(\#blocks)$ memory per process
- 2 system quiesence states **vs** $\#level$ reductions for mesh restructuring
- $O(1)$ **vs** $O(\log P)$ time neighbor lookup

[1] Langer, et.al. Scalable Algorithms for Distributed Memory Adaptive Mesh Refinement. In 24th International Conference on Computer Architecture and High Performance Computing (SBAC-PAD) 2012, NY, USA.

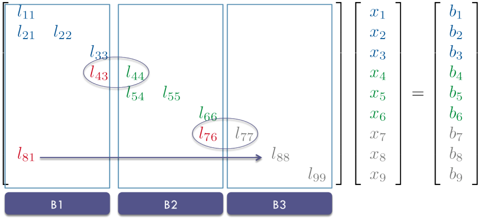
Adaptive Mesh Refinement



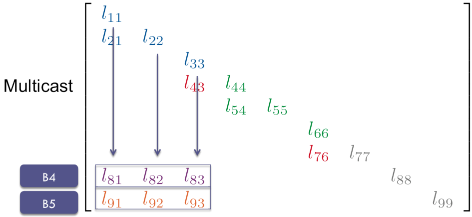
Timesteps per second strong scaling on IBM BG/Q with a max depth of 15.

Sparse Triangular Solver (sloc: 692)

Matrix Decomposition



Column Decomposition



Dense Parts Decomposition

- Methods express dependencies
  - Avoid MPI_Iprobe, MPI_Test etc
- Overdecomposition and round-robin mapping
  - overlap and load balance
- Dynamic creation of parallel units
  - Distributes dense matrix regions
- SLOC (692 vs 897 for SuperLU)
  - more sophisticated, higher performance algorithm

Sparse Triangular Solver

Parallel Algorithm

Schedule independent computation

High priority: Process messages

Low priority: do independent computation

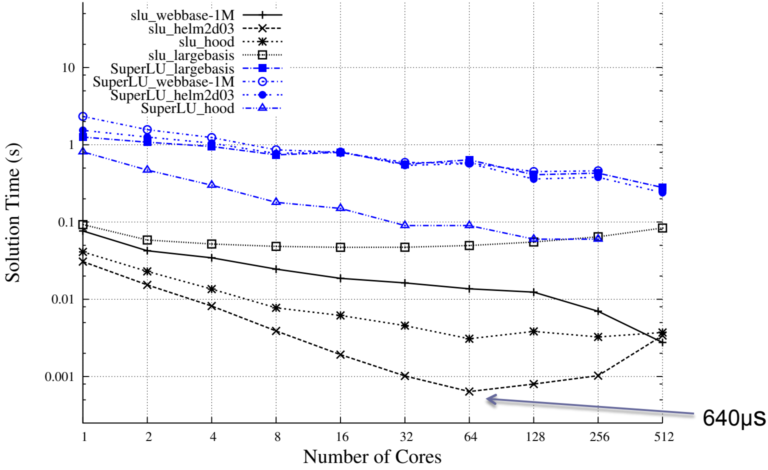
Highest priority: receive solution values

High priority: Process messages

```
if (onDiagonalChare) {
  serial { thisProxy[thisIndex].indepCompute(...) }
  overlap {
    while (!finished) {
      when recvData(int len, double data[len], int rows[len])
        serial {if(len>0) diagReceiveData(len, data, rows);}
    }
    when indepCompute(int a) serial {myIndepCompute();}
  }
} else {
  when getXvals(xValMsg* msg) serial {nondiag_compute();}
  while (!finished) {
    when recvData(int len, double data[len], int rows[len])
    serial {nondiagReceiveData(len, data, rows);}
  }
}
```

Sparse Triangular Solver

Performance



Charm++ and PPL at SC12

Charm++ Tutorial 8:30AM-12:00PM on Sunday November 11 (past)

HPC Challenge BoF 12:15PM-1:15PM on Tuesday November 13, in 255-A (you're here!)

Fernbach Award Talk 11:30AM-12:00PM on Wednesday November 14th, in 155-E

Doctoral Showcase 11:15AM - 11:30AM on Wednesday, in 155-F, by Osman Sarood "Saving Energy and Power"

Paper Presentation 2:00-2:30PM on Wednesday Nov 14, in 355-EF, Talk by Yanhua Sun "Optimizing fine-grained communication in a biomolecular dynamics simulation application on Cray XK6"

Charm++ BoF 12:15PM-1:15PM on Thursday November 15, in 255-A

For more info

http://charm.cs.illinois.edu/

LeanMD: Adding FT and LB

Trivial additions but major gains

```
serial { AtSync(); }
when ResumeFromSync() { }
```

LeanMD: Adding FT and LB

Trivial additions but major gains

```
serial { AtSync(); }
when ResumeFromSync() { }
```

```
if (stepCount % checkptFreq == 0) {
    serial {
        //coordinate to start checkpointing
        contribute(CkCallback(CkReductionTarget(Cell,startCheckpoint),thisProxy(0,0,0)));
    }
    if (thisIndex.x == 0 && thisIndex.y == 0 && thisIndex.z == 0) {
        when startCheckpoint() serial {
            CkCallback cb(CkReductionTarget(Cell,recvCheckPointDone), thisProxy);
            if (checkptStrategy == 0) CkStartCheckpoint(logs.c_str(), cb);
            else CkStartMemCheckpoint(cb);
        }
    }
    when recvCheckPointDone() { }
}
```

LeanMD: Adding FT and LB

Trivial additions but major gains

```
serial { AtSync(); }
when ResumeFromSync() { }
```

```
if (stepCount % checkptFreq == 0) {
    serial {
        //coordinate to start checkpointing
        contribute(CkCallback(CkReductionTarget(Cell,startCheckpoint),thisProxy(0,0,0)));
    }
    if (thisIndex.x == 0 && thisIndex.y == 0 && thisIndex.z == 0) {
        when startCheckpoint() serial {
            CkCallback cb(CkReductionTarget(Cell,recvCheckPointDone), thisProxy);
            if (checkptStrategy == 0) CkStartCheckpoint(logs.c_str(), cb);
            else CkStartMemCheckpoint(cb);
        }
    }
    when recvCheckPointDone() { }
}
```

```
//kill one of processes to demonstrate fault tolerance
if (stepCount == 30 && thisIndex.x == 1 && thisIndex.y == 1 && thisIndex.z == 0) serial {
    if (CkHasCheckpoints()) {
        CkDieNow();
    }
}
```