Toucan
A Translator for Communication Tolerant MPI Applications

Sergio Martin
*University of California
San Diego

Marsha J. Berger
Courant Institute
New York University

Scott B. Baden*
Lawrence Berkeley
National Laboratory

31st IEEE International Parallel & Distributed Processing Symposium

UC San Diego
JACOBS SCHOOL OF ENGINEERING
Computer Science and Engineering

NYU | COURANT INSTITUTE OF MATHEMATICAL SCIENCES

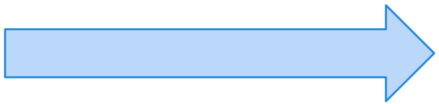
BERKELEY LAB

Motivation

- **Problem:** Communication costs are significant in large-scale parallel applications
  - Moreover, the overheads are continuing to grow towards the Exascale.
- **Coping strategies:**
  - Tolerate or avoid communication.
  - One Approach:
    Overlap communication with computation by manually restructuring the code.
- **Shortfalls:**
  - Entangles the overlap strategy with the application logic.
  - Requires a considerable amount of effort.
  - For large applications, this would be impractical.

Anatomy of a (Typical) MPI Program

```
Begin

    Initialize Data

    Main Loop
     {
       … Receives …
       … Sends …
       … Compute …
     }

     Other Communication
     Output Result

End
```




```
Begin

    Initialize Data

    Main Loop
     {
       … Receives …
       … Sends …
       … Compute(Independent) …
           -- Wait --
       … Compute(Dependent) …
     }

    Other Communication
    Output Result

End
```

Introducing **Toucan**

- A Source-to-Source Translator of C/C++ MPI Applications.
  - Automatically generates a communication-tolerant variant of the source code.
  - Guided by programmer annotations (directives).
  - Based on the annotation scheme of our previous work: **Bamboo**.
  - Built using the ROSE Compiler Framework (LLNL).
- The translated code is compiled/linked to execute with our runtime system: **MATE**.
  - MATE uses a dynamic scheduler that encapsulates most of the scheduling complexity in the runtime system.
  - This strategy avoid code bloating compared to static scheduling and inlining (Bamboo).
  - Supports recursive code.

(Pronounced ‘Mah-tay’)

Toucan's Approach

```
Begin

    Initialize Data

    Main Loop
     {
      … Receives …
      … Sends …
      … Compute …
     }

    Other Communication
    Output Result

End
```

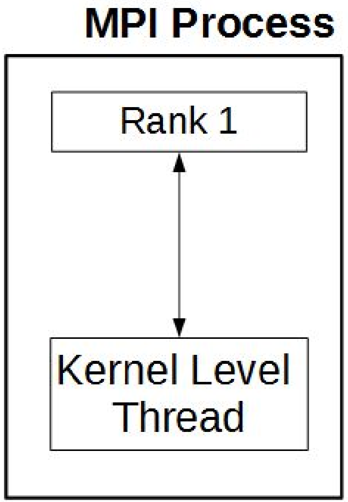
Programmer Annotates Code Regions Using Toucan Directives

```
Begin

    Initialize Data

    # Toucan Superblock
    Main Loop
     {
      { … Receives … } # Send Region
      { … Sends …} # Receive Region
      { … Compute … } # Compute Region
     }

    Other Communication
    Output Result

End
```

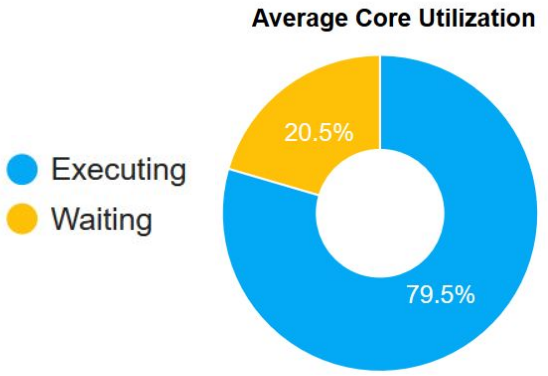
Limits to Overlap

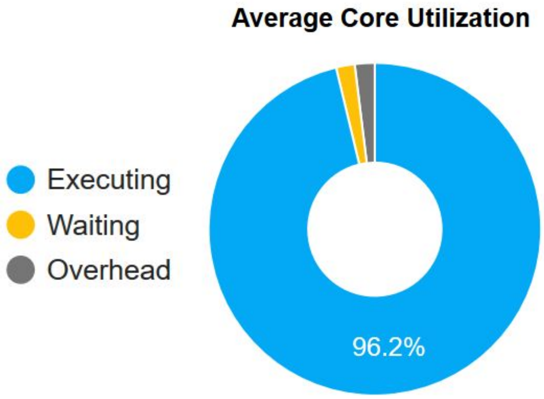
- In a typical MPI execution, each rank is assigned to a single thread.
  - Even with manually restructured codes, overlap is limited to a single rank/core.
  - Cores will sit idle while other ranks may be ready to execute.

- **Overdecomposition** can help improve overlap.
  - Idea: create multiple ranks for each core (AMPI / Charm++)
  - A core can switch to another rank while other are still waiting for communication operations.

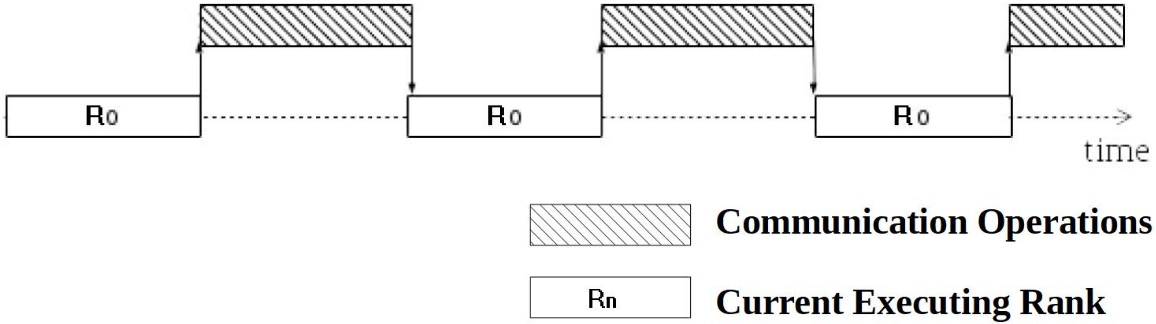


Core Usage Timeline

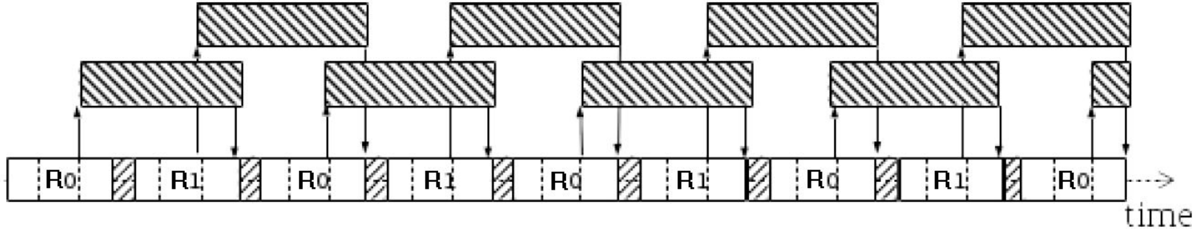




MPI - No Overlap



Toucan - 2 Subranks per Core



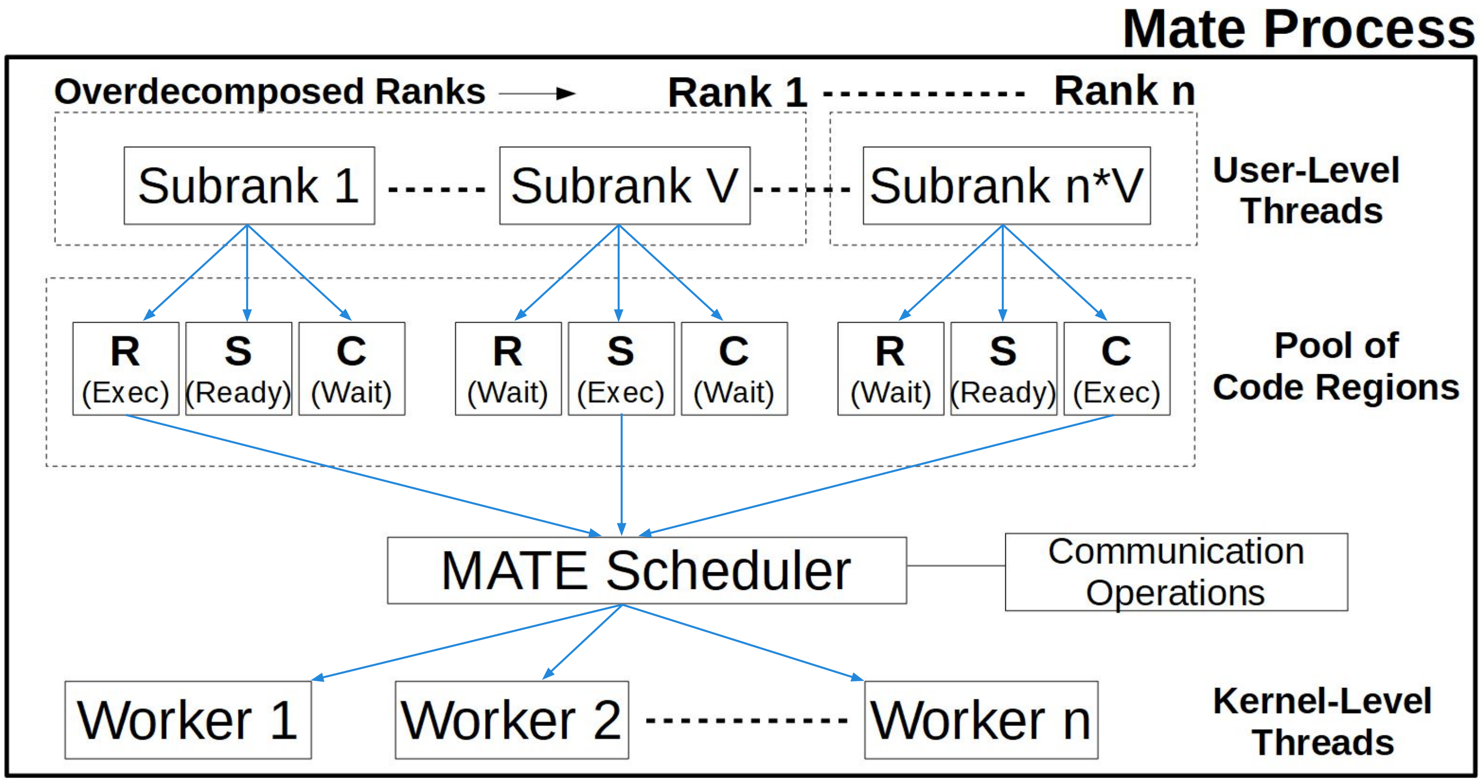
Output Code

Example: *1D Stencil Jacobi Solver*

```
#pragma toucan superblock
for (int iter = 0; iter <   K ; iter++) {
    #pragma toucan receive
    { MPI_Irecv(recvBuffer ← [left  neighbor]);
      MPI_Irecv(recvBuffer ← [right neighbor]); }

    #pragma toucan send
    { Pack(Uprev → leftSendBuffer);
      Pack(Uprev → rightSendBuffer);
      MPI_Isend(leftSendBuffer → [left  neighbor]);
      MPI_Isend(rightSendBuffer → [right neighbor]); }

    #pragma toucan compute
    { MPI_Waitall();
      Unpack(U ← leftSendBuffer);
      Unpack(U ← rightSendBuffer);
      for (int i = 0; i < N; i++)
            U[i] = Uprev[i−1] − 2∗Uprev[i] +  Uprev[i+1];
      swap(&U, &Uprev); }
}
```

```
int iter_0 = 0, iter_1 = 0, iter_2 = 0

while(MateGetNextRegion(&regionId))
 switch (regionId) {
  case NO_REGION_READY:
        Mate_SuspendSubRank();  break;

  case "receive":
      Mate_RequestData(recvBuffer ← [left  neighbor]);
      Mate_RequestData(recvBuffer ← [right neighbor]);
      iter_0++; if (iter_0 >= K) Mate_FinishRegion("receive");
      else Mate_AdvanceRegion("receive");
      break;

  case "send":
      Pack(Uprev → leftSendBuffer);
      Pack(Uprev → rightSendBuffer);
      Mate_PushData(sendBuffer → [left  neighbor]);
      Mate_PushData(sendBuffer → [right neighbor]);
      iter_1++; if (iter_1 >= K) Mate_FinishRegion("send");
      else Mate_AdvanceRegion("send");
      break;

  case "compute":
      // MPI_Waitall(); call elided during translation
      Unpack(U ← leftSendBuffer);
      Unpack(U ← rightSendBuffer);
      for (int i = 0; i < N; i++)
          U[i] = U[i−1] + U[i+1] − 2∗U[i];
      swap(&U, &Uprev);
      iter_2++; if (iter_2 >= K) Mate_FinishRegion("compute");
      else Mate_AdvanceRegion("compute");
      break;

  default: error_handler(); break;
 }
```

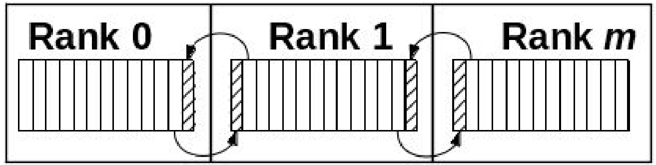
Toucan's Implementation



Annotating an MPI Program

Example: *1D Stencil Jacobi Solver*

```
for (int iter = 0; iter <   K  ; iter++) {

        MPI_Irecv(recvBuffer ← [left  neighbor]);
        MPI_Irecv(recvBuffer ← [right neighbor]);


        Pack(Uprev → leftSendBuffer);
        Pack(Uprev → rightSendBuffer);
        MPI_Isend(leftSendBuffer → [left  neighbor]);
        MPI_Isend(rightSendBuffer → [right neighbor]);


        MPI_Waitall();
        Unpack(U ← leftSendBuffer);
        Unpack(U ← rightSendBuffer);
        for (int i = 0; i < N; i++)
            U[i] = Uprev[i−1] − 2*Uprev[i] +  Uprev[i+1];
        swap(&U, &Uprev);
}
```



1D Grid (*N* elements)

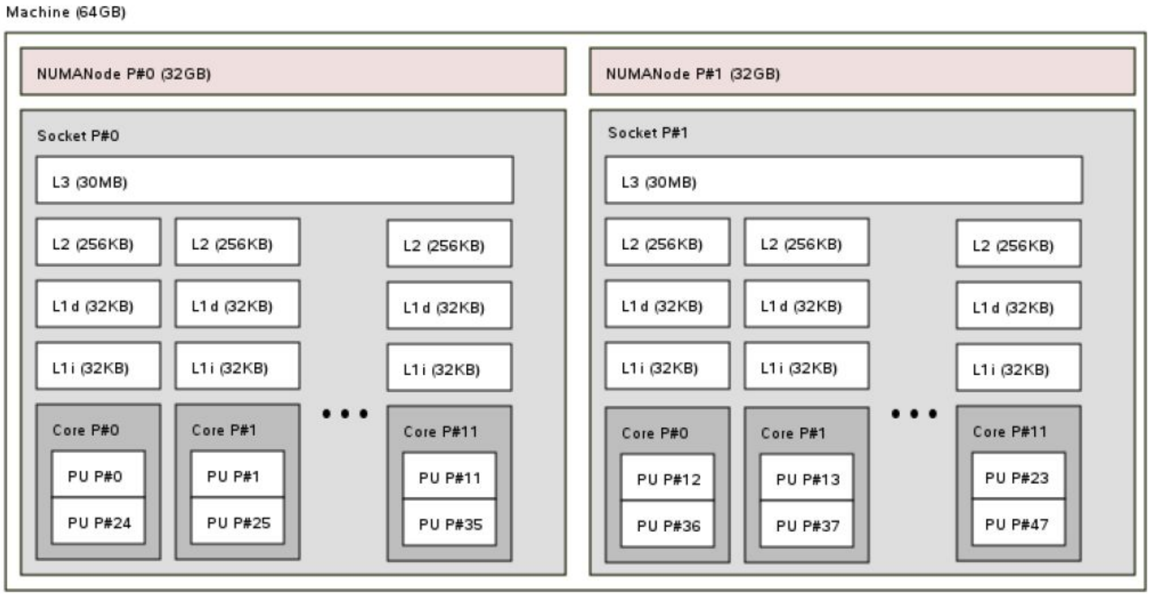
Hardware Testbed: Edison @ NERSC

Node Configuration:

- 2 x 12-core Intel *Ivy Bridge* processors (Total: 24 cores per Node) @ 2.4 Ghz

Memory:

- 1 NUMA Node per Processor
- 32 Gb DDR3 per NUMA
- 64 Gb DDR3 Total per Node



Software:

- Cray-MPICH v7.4.1
- Intel icc compiler 15.0.1 (-O3)
- Intel MKL Library (for *dgemm*)

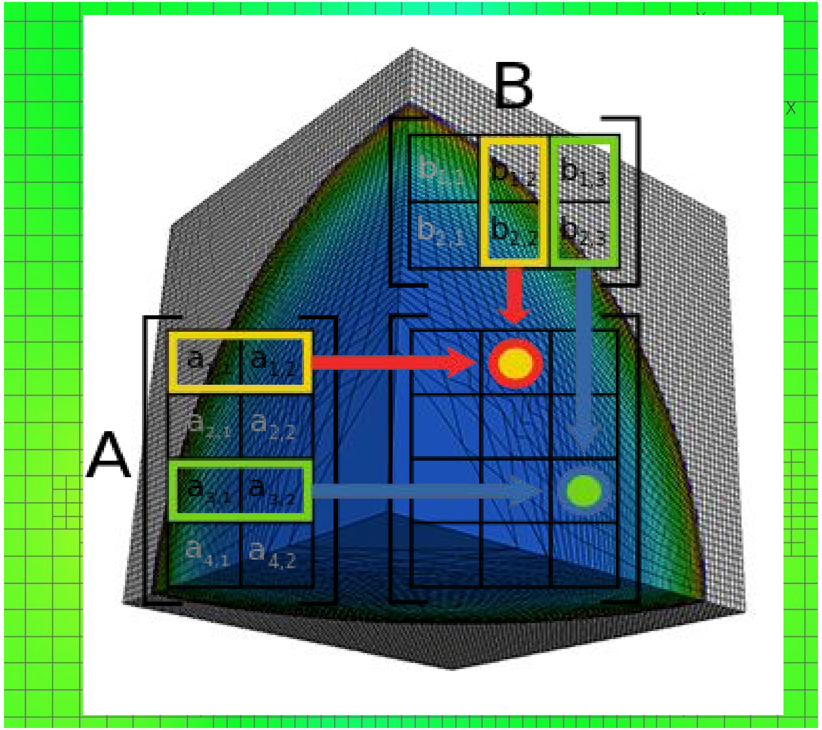
Test Cases

Algorithms:

- Cannon 2D (Linear Algebra)
- LULESH 2.0 (Unstructured Grid)
- Mpix_FlowCart (Unstructured MG)

Code Variants:

- MPI Original (Base Case)
- Manually Overlapped*
- Toucan
- Ideal (No communication)



Source: NASA Ames Research Center

Dense Linear Algebra

Test Case: 2D Cannon's Algorithm

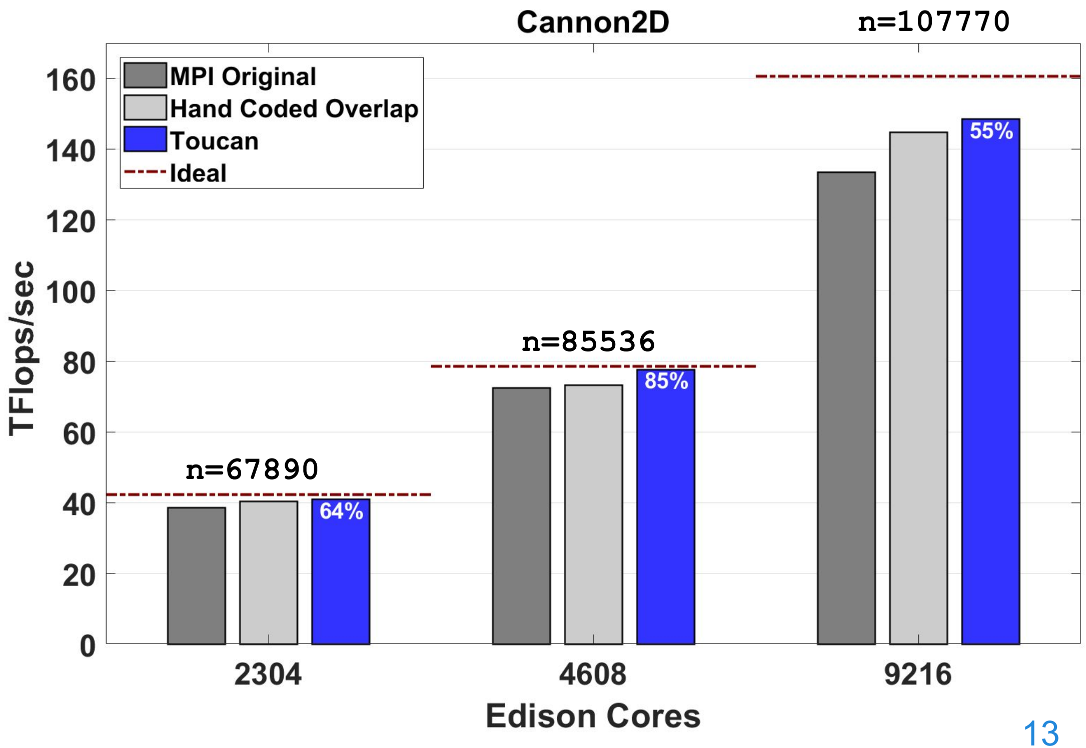
Computes the matrix product of two matrices in a series of $\sqrt{P}$ steps, where P = # of ranks. Each step rotates blocks of matrices A and B along rows and columns of the 2D rank array.

Hand Coded Overlap employs additional buffers while computing the next step.

4 directives required for Toucan Version

Experiment Details:
Weak Scaling Study (Flops/core constant).

Results:
64 to 85% Communication Hidden by Toucan



Unstructured Grid (Regular)

Test Case: LULESH 2.0

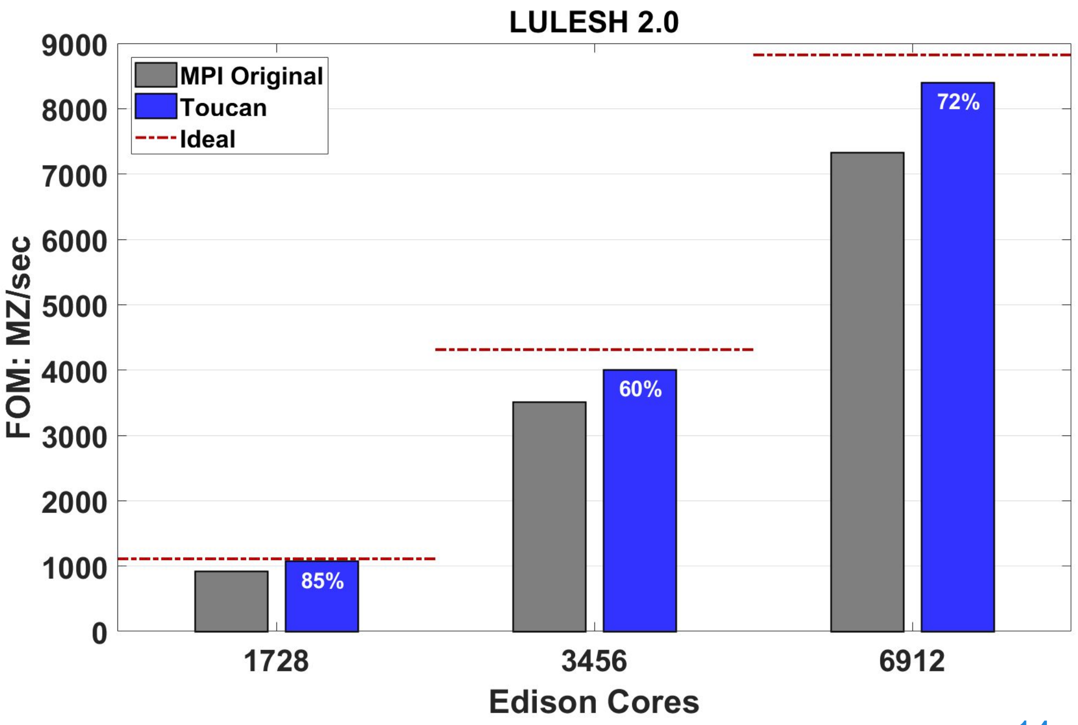
LULESH is highly simplified hydrodynamics application, developed as a proxy application at the Lawrence Livermore National Lab.

No Manually Overlapped variant available.

20 directives required for Toucan Version (5 Superblocks)

Experiment Details:
Weak Scaling Study (64^3 elements/core).

Results:
60 to 85% Communication Hidden by Toucan



Unstructured Grid (irregular)

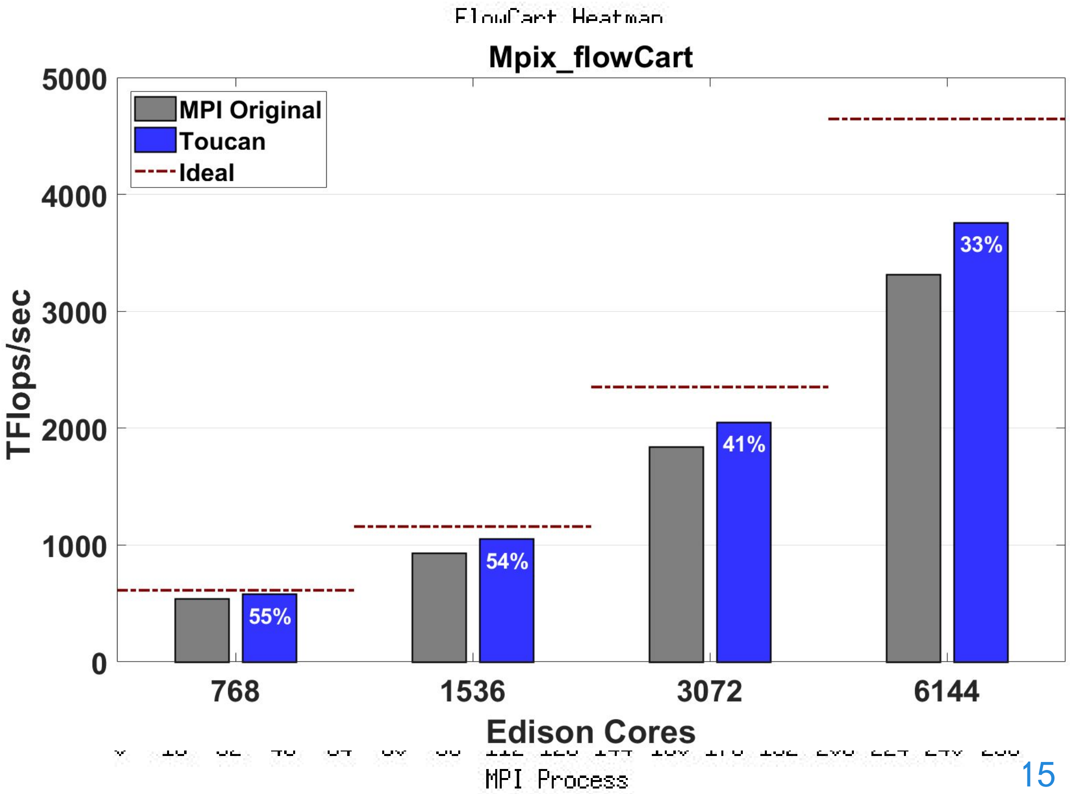
Test Case: Mpix_flowCart

Production code developed by NASA Ames. Uses multigrid with an irregular mesh to solve the compressible Euler equations. Used in Aerospace design, hundreds of users.

No Manually Overlapped variant available (20K lines of code). 20 directives required for Toucan Version (4 Superblocks)

Experiment Details:
Strong Scaling Study (75M Cell Mesh).

Results:
33 to 55% Communication Hidden by Toucan



Conclusions

- Toucan was able to hide between **33%** and **85%** of the communication cost in 3 common HPC application motifs.
- Only a modest amount of annotation was required.
- Dynamic Scheduling was key to avoiding code bloating and support recursion.
- **Limitations:**
  - Collective communication operations are not overlapped by Toucan.
  - Global and static variables need to be privatized.
- **Current/Future Steps:**
  - Investigate performance fall off in Mpix_flowCart.
  - Generalize the RSC model to a broader annotation model.
  - Create a hybrid model where local tasks can communicate through SHMEM.

Related Work

- **Bamboo:**
  - “Bamboo - Translating MPI Applications to a Latency-tolerant, Data-driven Form”
    T. Nguyen, P. Cicotti, E. Bylaska, D. Quinlan, and S. Baden. In: Supercomputing ‘12.
- **Adaptive MPI:**
  - “Adaptive MPI”. C. Huang, O. Lawlor, and L. V. Kalé. In: LCPC ‘04.
- **MPI/SMPSs:**
  - “Overlapping Communication and Computation by Using a Hybrid MPI/SMPSs Approach”
    V. Marjanović, J. Labarta, E. Ayguadé, and M. Valero . In: ICS ‘10.
- **Delta Send-Recv Model:**
  - “Overlapping Communication and Computation by Using a Hybrid MPI/SMPSs Approach”
    B. Bao, C. Ding, Y. Gao, and R. Archambault. In: CCGRID '12.
- **Compiler based techniques.**
  - “Exact Dependence Analysis for Increased Communication Overlap”
    S. Pellegrini, T. Hoefler, and T. Fahringer. In: EuroMPI ’12
  - “MPI-aware Compiler Optimizations for Improving Communication-computation Overlap”
    A. Danalis, L. Pollock, M. Swany, J. Cavazos. In: ICS ‘09

Questions?

Website:

mate.ucsd.edu/toucan

Contact:

sergiom@eng.ucsd.edu
berger@cims.nyu.edu
baden@lbl.gov