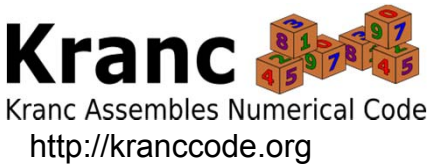
Automated Code Generation and High-Level Code Optimization in the Cactus Framework

Ian Hinder, Erik Schnetter,
Gabrielle Allen, Steven R. Brandt, and Frank Löffler



http://cactuscode.org

LSU
CENTER FOR COMPUTATION & TECHNOLOGY

The Cactus Framework

- Component Based Software Engineering Framework
- Metaphor:
  - Central infrastructure: The Flesh
  - Programmer defined modules: Thorns
- General purpose toolkit, but created to support General Relativity
- Originally created by Paul Walker and Joan Masso at the Albert Einstein Institute

Kranc
Kranc Assembles Numerical Code
http://kranccode.org

http://cactuscode.org

LSU
CENTER FOR COMPUTATION & TECHNOLOGY

The Cactus Framework

- Schedule Tree
  - Sequential workflow with steps that run in parallel
  - Starts with a predefined set of steps for initialization, pre-evolve, evolve, post-evolve, etc.
- Groups
  - Programmer-defined step that can be added to the schedule tree

Kranc
Kranc Assembles Numerical Code
http://kranccode.org

http://cactuscode.org

LSU
CENTER FOR COMPUTATION & TECHNOLOGY

The Cactus Framework

- Grid Functions
  - Multi-Dimensional Arrays
  - Span Multiple Compute Nodes
  - Synchronized through Ghost Zones and/or Interpolation, Managed by Driver Layer
- Cactus Configuration files (CCL)
  - Domain specific configuration language that describes groups, grid functions, input parameters, etc.

Kranc
Kranc Assembles Numerical Code
http://kranccode.org

http://cactuscode.org

LSU
CENTER FOR COMPUTATION & TECHNOLOGY

The Cactus Framework

- Supports C, C++, F77, F90
- Programmer does not have to worry about parallelism or mesh refinement, the abstraction enables parallel Adaptive Mesh Refinement or unigrid code that scales to thousands of cores
- Checkpoint/Restart to other architectures/core counts
- Simulation control, data analysis, visualization support
- Active user community, 13+ years old
- Enables collaborative development

Kranc
Kranc Assembles Numerical Code
http://kranccode.org

http://cactuscode.org

LSU
CENTER FOR COMPUTATION & TECHNOLOGY

Kranc

Kranc Assembles Numerical Code

- A tool for generating thorns
- Turns Mathematica equations into code + Cactus configuration files
- Original version written by S. Husa, Ian Hinder, C. Lechner at AEI and Southampton
- Used for many scientific papers
- Allows user to specify continuum problem without concentrating on numerical implementation

LSU
CENTER FOR COMPUTATION & TECHNOLOGY

Kranc

Kranc Assembles Numerical Code

- Minimally, a Kranc script contains
  - Cactus Group Definitions
  - A set of Calculations
- A calculation is a Kranc data structure which describes how to update a set of grid functions from other grid functions.
- String identifies schedule group.

LSU
CENTER FOR COMPUTATION & TECHNOLOGY

Kranc Features

- Tensorial input checked for index consistency
  - dot[h[la,lb]] -> -2 alpha K[la,lb].
- Calculation can be performed on Interior, Boundary, or Everywhere.
- Allows simple boundary conditions to be coded.
- Generated thorn can inherit from other thorns
- http://kranccode.org

LSU
CENTER FOR COMPUTATION & TECHNOLOGY

DSL For Kranc

- Mathematica also presents some difficulties...
  - Would like to use more mathematical notation: G^a_bc instead of G[ua,lb,lc]
  - Error messages from Mathematica are obscure and reproduce all of Kranc input
  - Small parsing expression grammar framework can help here

Kranc Assembles Numerical Code
http://kranccode.org

http://cactuscode.org

LSU
CENTER FOR COMPUTATION & TECHNOLOGY

Input File for Wave Eqn

```
@THORN SimpleWave
@DERIVATIVES
  PDstandard2nd[i_] -> StandardCenteredDifferenceOperator[1,1,i],
  PDstandard2nd[i_, i_] -> StandardCenteredDifferenceOperator[2,1,i],
  PDstandard2th[i_, j_] -> StandardCenteredDifferenceOperator[1,1,i] *
                           StandardCenteredDifferenceOperator[1,1,j]
@END_DERIVATIVES
@TENSORS
  phi, pi
@END_TENSORS
@GROUPS
  phi -> "phi_group",
  pi -> "pi_group"
@END_GROUPS
@DEFINE PD = PDstandard2nd
...
```

```
...
@CALCULATION "initial_sine"
  @Schedule {"AT INITIAL"}
  @EQUATIONS
    phi -> Sin[2 Pi (x - t)],
    pi -> -2 Pi Cos[2 Pi (x - t)]
  @END_EQUATIONS
@END_CALCULATION
@CALCULATION "calc_rhs"
  @Schedule {"in MoL_CalcRHS"}
  @EQUATIONS
    dot[phi] -> pi,
    dot[pi]  -> Euc[ui,uj] PD[phi,li,lj]
  @END_EQUATIONS
@END_CALCULATION
@END_THORN
```

Kranc
Kranc Assembles Numerical Code
http://kranccode.org

http://cactuscode.org

LSU
CENTER FOR COMPUTATION & TECHNOLOGY

The Einstein Equations

$$G_{\mu\nu} = 8\pi T_{\mu\nu}$$

It looks so simple... it's just 10 coupled non-linear partial differential equations

Kranc
Kranc Assembles Numerical Code
http://kranccode.org

http://cactuscode.org

The Einstein Eqns in Mathematica

```
dir[ua] -> Sign[beta[ua]],

epsdiss[ua] -> EpsDiss,

detgt -> 1,

gtu[ua,ub]   -> 1/detgt detgtExpr MatrixInverse [gt[ua,ub]],
Gtl[la,lb,lc]  -> 1/2
            (PD[gt[lb,la],lc] + PD[gt[lc,la],lb] - PD[gt[lb,lc],la]),
Gtlu[la,lb,uc] -> gtu[uc,ud] Gtl[la,lb,ld],
Gt[ua,lb,lc]   -> gtu[ua,ud] Gtl[ld,lb,lc],

Xtn[ui] -> gtu[uj,uk] Gt[ui,lj,lk],

Rt[li,lj] -> - (1/2) gtu[ul,um] PD[gt[li,lj],ll,lm]
           + (1/2) gt[lk,li] PD[Xt[uk],lj]
           + (1/2) gt[lk,lj] PD[Xt[uk],li]
           + (1/2) Xtn[uk] Gtl[li,lj,lk]
           + (1/2) Xtn[uk] Gtl[lj,li,lk]
           + (+ Gt[uk,li,ll] Gtlu[lj,lk,ul]
             + Gt[uk,lj,ll] Gtlu[li,lk,ul]
             + Gt[uk,li,ll] Gtlu[lk,lj,ul]),

fac1 -> IfThen [conformalMethod, -1/(2 phi), 1],
cdphi[la] -> fac1 CDt[phi,la],
fac2 -> IfThen [conformalMethod, 1/(2 phi^2), 0],
cdphi2[la,lb] -> fac1 CDt[phi,la,lb] + fac2 CDt[phi,la] CDt[phi,lb],

Rphi[li,lj] -> - 2 cdphi2[lj,li]
           - 2 gt[li,lj] gtu[ul,un] cdphi2[ll,ln]
           + 4 cdphi[li] cdphi[lj]
           - 4 gt[li,lj] gtu[ul,un] cdphi[ln] cdphi[ll],

Atm[ua,lb] -> gtu[ua,uc] At[lc,lb],
Atu[ua,ub] -> Atm[ua,lc] gtu[ub,uc],

e4phi      -> IfThen [conformalMethod, 1/phi^2, Exp[4 phi]],
em4phi     -> 1 / e4phi,
g[la,lb]   -> e4phi gt[la,lb],
detg       -> detgExpr,
```

```
gu[ua,ub]   -> em4phi gtu[ua,ub],

R[la,lb] -> Rt[la,lb] + Rphi[la,lb],

(* Matter terms *)

rho -> addMatter
      (1/alpha^2 (T00 - 2 beta[ui] T0[li] + beta[ui] beta[uj] T[li,lj])),

S[li] -> addMatter (-1/alpha (T0[li] - beta[uj] T[li,lj])),

trS -> addMatter (em4phi gtu[ui,uj] T[li,lj]),

(* RHS terms *)

dot[phi]      -> IfThen [conformalMethod, 1/3 phi, -1/6] alpha trK
               + Upwind[beta[ua], phi, la]
               + epsdiss[ua] PDdiss[phi,la]
               + IfThen [conformalMethod, -1/3 phi, 1/6] PD[beta[ua],la],

dot[gt[la,lb]] -> - 2 alpha At[la,lb]
               + Upwind[beta[uc], gt[la,lb], lc]
               + epsdiss[uc] PDdiss[gt[la,lb],lc]
               + gt[la,lc] PD[beta[uc],lb] + gt[lb,lc] PD[beta[uc],la]
               - (2/3) gt[la,lb] PD[beta[uc],lc],
dotXt[ui]     -> - 2 Atu[ui,uj] PD[alpha,lj]
               + 2 alpha (+ Gt[ui,lj,lk] Atu[uk,uj]
                          - (2/3) gtu[ui,uj] PD[trK,lj]
                          + 6 Atu[ui,uj] cdphi[lj])
               + gtu[uj,ul] PD[beta[ui],lj,ll]
               + (1/3) gtu[ui,uj] PD[beta[ul],lj,ll]
               + Upwind[beta[uj], Xt[ui], lj]
               + epsdiss[uj] PDdiss[Xt[ui],lj]
               - Xtn[uj] PD[beta[ui],lj]
               + (2/3) Xtn[ui] PD[beta[uj],lj]
               + addMatter (- 16 pi alpha gtu[ui,uj] S[lj]),
dot[Xt[ui]]   -> dotXt[ui],

dottrK        -> - em4phi ( gtu[ua,ub] ( PD[alpha,la,lb]
                          + 2 cdphi[la] PD[alpha,lb] )
                          - Xtn[ua] PD[alpha,la] )
               + alpha (Atm[ua,lb] Atm[ub,la] + (1/3) trK^2)
               + Upwind[beta[ua], trK, la]
               + epsdiss[ua] PDdiss[trK,la]
               + addMatter (4 pi alpha (rho + trS)),
dot[trK]      -> dottrK,
```

```
Ats[la,lb]     -> - CDt[alpha,la,lb] +
                  + 2 (PD[alpha,la] cdphi[lb] + PD[alpha,lb] cdphi[la] )
                  + alpha R[la,lb],
trAts          -> gu[ua,ub] Ats[la,lb],
dot[At[la,lb]] -> + em4phi (+ Ats[la,lb] - (1/3) g[la,lb] trAts )
                  + alpha (trK At[la,lb] - 2 At[la,lc] Atm[uc,lb])
                  + Upwind[beta[uc], At[la,lb], lc]
                  + epsdiss[uc] PDdiss[At[la,lb],lc]
                  + At[la,lc] PD[beta[uc],lb] + At[lb,lc] PD[beta[uc],la]
                  - (2/3) At[la,lb] PD[beta[uc],lc]
                  + addMatter (- em4phi alpha 8 pi
                               (T[la,lb] - (1/3) g[la,lb] trS)),

dot[alpha] -> - harmonicF alpha^harmonicN
              (+ LapseACoeff       A
               + (1 - LapseACoeff) trK)
             + LapseAdvectionCoeff Upwind[beta[ua], alpha, la]
             + epsdiss[ua] PDdiss[alpha,la],

dot[A]     -> + LapseACoeff (dottrK - AlphaDriver A)
              + LapseAdvectionCoeff Upwind[beta[ua], A, la]
              + epsdiss[ua] PDdiss[A,la],

eta -> etaExpr,
theta -> thetaExpr,

dot[beta[ua]] -> + theta ShiftGammaCoeff
                 (+ ShiftBCoeff       B[ua]
                  + (1 - ShiftBCoeff) (Xt[ua] - eta BetaDriver beta[ua]))
                + ShiftAdvectionCoeff Upwind[beta[ub], beta[ua], lb]
                + epsdiss[ub] PDdiss[beta[ua],lb],

dot[B[ua]]    -> + ShiftBCoeff (dotXt[ua] - eta BetaDriver B[ua])
                 + ShiftAdvectionCoeff Upwind[beta[ub], B[ua], lb]
                 - ShiftAdvectionCoeff Upwind[beta[ub], Xt[ua], lb]
                 + epsdiss[ub] PDdiss[B[ua],lb]
```

http://kranccode.org

http://cactuscode.org

The Einstein Equations in C

```
CCTK_REAL rho = INV(SQR(alphaL))*(eTttL - 2*(beta2L*eTtyL +
  beta3L*eTtzL) + 2*(beta1L*(-eTtxL + beta2L*eTxyL + beta3L*eTxzL) +
  beta2L*beta3L*eTyzL) + eTxxL*SQR(beta1L) + eTyyL*SQR(beta2L) +
  eTzzL*SQR(beta3L));

CCTK_REAL S1 = (-eTtxL + beta1L*eTxxL + beta2L*eTxyL +
  beta3L*eTxzL)*INV(alphaL);

CCTK_REAL S2 = (-eTtyL + beta1L*eTxyL + beta2L*eTyyL +
  beta3L*eTyzL)*INV(alphaL);

CCTK_REAL S3 = (-eTtzL + beta1L*eTxzL + beta2L*eTyzL +
  beta3L*eTzzL)*INV(alphaL);

CCTK_REAL trS = em4phi*(eTxxL*gtu11 + eTyyL*gtu22 + 2*(eTxyL*gtu12 +
  eTxzL*gtu13 + eTyzL*gtu23) + eTzzL*gtu33);

CCTK_REAL phirhsL = epsdiss1*PDdissipationNth1phi +
  epsdiss2*PDdissipationNth2phi + epsdiss3*PDdissipationNth3phi +
  beta1L*PDupwindNthAnti1phi + beta2L*PDupwindNthAnti2phi +
  beta3L*PDupwindNthAnti3phi + PDupwindNthSymm1phi*Abs(beta1L) +
  PDupwindNthSymm2phi*Abs(beta2L) +...

/* Plus a little more than a thousand lines ... */
```

Kranc

Kranc Assembles Numerical Code

http://kranccode.org

http://cactuscode.org

LSU
CENTER FOR COMPUTATION & TECHNOLOGY

The Einstein Equations

- It's one kernel
- Split in two parts to avoid instruction cache misses
- Hard for compilers to optimize/vectorize
- Examined assembly output
  - No fission, blocking, unrolling, vectorization, ...
  - No useful compiler reports either

Kranc
Kranc Assembles Numerical Code
http://kranccode.org

http://cactuscode.org

LSU
CENTER FOR COMPUTATION & TECHNOLOGY

$G_{\mu\nu}$ = Slow!

- All modern CPUs (Intel, AMD, IBM Blue Gene, IBM Power, even GPUs) execute floating point instructions in vector units
- Typically operating on 2 or 4 values simultaneously (GPUs: up to 32 values, “warp”)
- Scalar code just throws away 50% or more of TPP (theoretical peak performance)

Kranc
Kranc Assembles Numerical Code
http://kranccode.org

http://cactuscode.org

LSU
CENTER FOR COMPUTATION & TECHNOLOGY

Alternative Approach: Use Kranc

- We already use Automated Code Generation (ACG) to generate our loop kernels
  - Actually, to generate complete Cactus components
- Many advantages: reduce code size, ensure correctness, easier experimentation and modification, …
- ***Also***: can apply high-level and low-level optimizations that compilers do not know about
  - General or domain-specific optimizations
  - Cannot always wait for optimization-related bug fixes to compilers and adding a stage to ACG framework is not too complex

Kranc
Kranc Assembles Numerical Code
http://kranccode.org

http://cactuscode.org

LSU
CENTER FOR COMPUTATION & TECHNOLOGY

Vectorization Design

- High-level architecture-independent API for vectorized code
- Provides implementations for various architectures
  - Intel/AMD SSE2, Intel/AMD AVX, Blue Gene “Double Hummer”, Power 7 VSX (as well as trivial scalar pseudo-vectorization)
- Vectorized code needs to be modified for:
  - Data types, load/store operations, arithmetic operations, loop strides
  - API imposes restrictions; not all scalar code can be easily vectorized
  - Memory alignment may require array padding for higher performance

Kranc
Kranc Assembles Numerical Code
http://kranccode.org

http://cactuscode.org

CENTER FOR COMPUTATION & TECHNOLOGY

Vectorization Implementation

- First attempt used C++, templates, and inline functions to ensure readable code
  - Failure: very slow when debugging, many compiler segfaults (GCC and Intel are most stable compilers)
- Now using macros defined in LSUThorns/Vectors (part of the EinsteinToolkit release called "Curie"):
  - Does not look as nice, e.g. **add(x, mul(y, z))**
- Targeting mostly stencil-based loop kernels
- All Kranc codes can benefit immediately
- Capabilities of different architectures surprisingly similar
  - Exceptions: unaligned memory access, partial vector load/store

CENTER FOR COMPUTATION & TECHNOLOGY

Vectorization Implementation

- Break up code into vectors

  $w_j = u_j + v_j$

  becomes

  $\{w_j, w_{j+1}\} = \{u_j, u_{j+1}\} + \{v_j, v_{j+1}\}$

- Mathematica transformation rules are applied

  e.g. xx_ * yy_ -> kmul[xx,yy]

Kranc
Kranc Assembles Numerical Code
http://kranccode.org

http://cactuscode.org

Vectorization API Functionality

- Aligned load, unaligned load
  - Compile-time logic to avoid unnecessary unaligned loads when accessing multi-D array elements
- Aligned store, partial vector store, store while bypassing cache
  - Load/store API not used symmetrically in applications!
  - Using partial vector operations at loop boundaries (instead of scalar operations) to avoid fixup code
- Arithmetic operations, including abs/max/min/sqrt/?:/etc.
  - Also including fma (fused multiply-add) operations explicitly, since compilers don't seem to automatically generate fma instructions for vectors...

LSU
CENTER FOR COMPUTATION & TECHNOLOGY

Performance Numbers

- Performance improvements depend on many details, in particular (for our code) i-cache misses
- Theoretical improvement: factor 2

System	CPU	Compiler	Speedup
Bethe	Intel	Intel	1.07
Blue Drop	Power 7	IBM	[NDA]
Curry	Intel	GCC	2.20 (?)
Hopper	AMD	GCC	1.47
Kraken	AMD	Intel	2.17 (?)
Ranger	AMD	Intel	1.41

Kranc
Kranc Assembles Numerical Code
http://kranccode.org

LSU
CENTER FOR COMPUTATION & TECHNOLOGY

Automated Code Gen with Cactus/Kranc

- Provides manageable high-level interface to equations
- Improves programmability
- Provides opportunities for specialized optimizations
- Reduces boilerplate

Kranc
Kranc Assembles Numerical Code
http://kranccode.org

http://cactuscode.org