EIN notation in the Diderot compiler

Charisee Chiw

University of Chicago

March 13, 2014

RoadMap

- Diderot
- Direct-Style Compiler
- Design of EIN notation
- Implementation
- Related and Future Work

Joint work with John Reppy, Gordon Kindlmann, Lamont Samuels, and Nick Seltzer.

RoadMap

- Diderot
- Direct-Style Compiler
- Design of EIN notation
- Implementation
- Related and Future Work

Joint work with John Reppy, Gordon Kindlmann, Lamont Samuels, and Nick Seltzer.

Diderot

- Motivation
  - Tools used to extract structure from image data
  - Creating new programs becomes part of the experimental process
- Diderot is parallel domain-specific programming language for scientific visualization and image analysis.
- Diderot supports a high-level of model of computation based on continuous tensor fields.

Diderot

- Motivation
  - Tools used to extract structure from image data
  - Creating new programs becomes part of the experimental process
- Diderot is parallel domain-specific programming language for scientific visualization and image analysis.
- Diderot supports a high-level of model of computation based on continuous tensor fields.

Diderot

- Motivation
  - Tools used to extract structure from image data
  - Creating new programs becomes part of the experimental process
- Diderot is parallel domain-specific programming language for scientific visualization and image analysis.
- Diderot supports a high-level of model of computation based on continuous tensor fields.

Image Analysis and Visualization

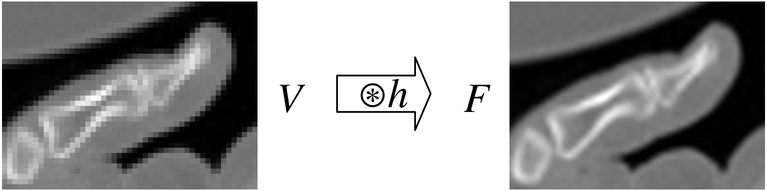
- We are interested in a class of algorithms that compute geometric properties of objects from imaging data.
- These algorithms compute over a continuous tensor field F (and its derivatives), which are reconstructed from discrete data using a separable convolution kernel h:

$$F = V \circledast h$$

Image Analysis and Visualization

- We are interested in a class of algorithms that compute geometric properties of objects from imaging data.
- These algorithms compute over a continuous tensor field F (and its derivatives), which are reconstructed from discrete data using a separable convolution kernel h:

$$F = V \circledast h$$



Discrete image data *Continuous field*

Tensor Fields

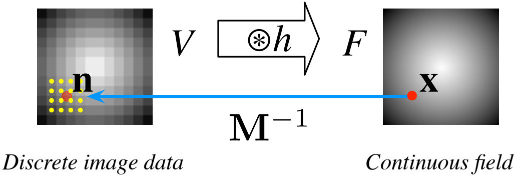
- Fields are functions from $\Re^d$ to tensors.
- Field types describe the domain, range, and continuity of the function:

$$\texttt{field\#}\overset{\text{levels of continuity}}{k}(\underset{\text{dimension of domain}}{d})\underbrace{[d_1, \ldots, d_n]}_{\text{shape of range}}$$

- Tensor Operators $(+, -, *, /, \cdot, :, \odot, \otimes, \times, trace, transpose, identity)$
- Tensor Operators lifted to work on fields $(+, -, *, /)$
- Higher order operations on fields $(\nabla, \nabla\cdot, \nabla\times, \nabla\otimes)$

Image Analysis and Visualization

A field application $F(\mathbf{x})$ gets compiled down into code that maps the world-space coordinates to image space and then convolves the image values in the neighborhood of the position.



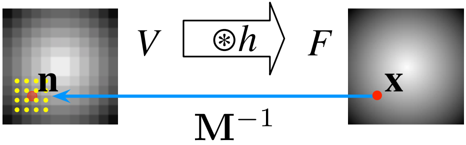
In 2D, the reconstruction is

$$F(\mathbf{x}) = \sum_{i=1-s}^{s} \sum_{j=1-s}^{s} V[\mathbf{n} + \langle i, j \rangle] h(\mathbf{f}_x - i) h(\mathbf{f}_y - j)$$

where s is the support of h, $\mathbf{n} = \lfloor \mathbf{M}^{-1}\mathbf{x} \rfloor$ and $\mathbf{f} = \mathbf{M}^{-1}\mathbf{x} - \mathbf{n}$.

Image Analysis and Visualization

A field application $F(\mathbf{x})$ gets compiled down into code that maps the world-space coordinates to image space and then convolves the image values in the neighborhood of the position.



Discrete image data *Continuous field*

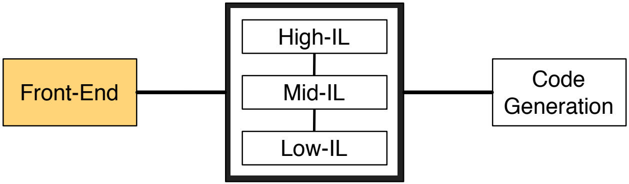
In 2D, the reconstruction is

$$F(\mathbf{x}) = \sum_{i=1-s}^{s} \sum_{j=1-s}^{s} V[\mathbf{n} + \langle i, j \rangle] h(\mathbf{f}_x - i) h(\mathbf{f}_y - j)$$

where s is the support of h, $\mathbf{n} = \lfloor \mathbf{M}^{-1}\mathbf{x} \rfloor$ and $\mathbf{f} = \mathbf{M}^{-1}\mathbf{x} - \mathbf{n}$.

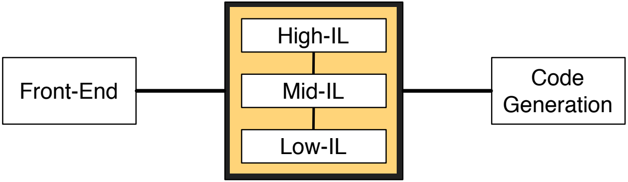
Compiler

- Front-end Phase
- Optimization and Lowering Phase
- Code Generation Phase



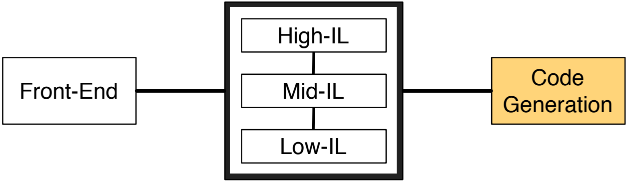
Compiler

- Front-end Phase
- Optimization and Lowering Phase
- Code Generation Phase



Compiler

- Front-end Phase
- Optimization and Lowering Phase
- Code Generation Phase



Current Compiler Limitations

- Support for additional operations on the surface language require many intermediate operators and normalization rules. $\nabla\times, \nabla\cdot$
- Require case-by-case analysis for normalization
- Generality versus implementation

Direct-Style operators for the inner product: "dot", "MulVecMat", "MulMatVec", "MulMatMat","MulVecTen3", "MulTen3Vec."

Contributions

Present the design of the new intermediate representation that we call EIN

- EIN is a concise way of represent tensor and field operators
- Implementation used for the current tensor and field operators
- Provide framework for richer set of operators to extend tensor operators up to fields $F \cdot G$
- Examine structure and do optimizations with a mathematically well-founded compiler
- General way to do code generation

Contributions

Present the design of the new intermediate representation that we call EIN

- EIN is a concise way of represent tensor and field operators
- Implementation used for the current tensor and field operators
- Provide framework for richer set of operators to extend tensor operators up to fields $F \cdot G$
- Examine structure and do optimizations with a mathematically well-founded compiler
- General way to do code generation

Contributions

Present the design of the new intermediate representation that we call EIN

- EIN is a concise way of represent tensor and field operators
- Implementation used for the current tensor and field operators
- Provide framework for richer set of operators to extend tensor operators up to fields $F \cdot G$
- Examine structure and do optimizations with a mathematically well-founded compiler
- General way to do code generation

Contributions

Present the design of the new intermediate representation that we call EIN

- EIN is a concise way of represent tensor and field operators
- Implementation used for the current tensor and field operators
- Provide framework for richer set of operators to extend tensor operators up to fields $F \cdot G$
- Examine structure and do optimizations with a mathematically well-founded compiler
- General way to do code generation

Contributions

Present the design of the new intermediate representation that we call EIN

- EIN is a concise way of represent tensor and field operators
- Implementation used for the current tensor and field operators
- Provide framework for richer set of operators to extend tensor operators up to fields $F \cdot G$
- Examine structure and do optimizations with a mathematically well-founded compiler
- General way to do code generation

Contributions

Present the design of the new intermediate representation that we call EIN

- EIN is a concise way of represent tensor and field operators
- Implementation used for the current tensor and field operators
- Provide framework for richer set of operators to extend tensor operators up to fields $F \cdot G$
- Examine structure and do optimizations with a mathematically well-founded compiler
- General way to do code generation

EIN Operators

Mathematicians have developed a notation for manipulating compact tensor expressions.
This inspired Einstein index notation (EIN).
The direct-style compiler used a fixed set of operators but we are replacing them with an EIN operator.
A new operator,

$$\lambda(\overline{T}).\langle e \rangle_{\alpha}$$

with EIN expression e, where $\overline{T}$ are parameters and α is the shape of the result.

EIN expressions

We use notation $\langle e \rangle_\alpha$ to represent an EIN expression e with tensor shape α. Subscripts can be bound to the outside of the EIN Expression as

$$\langle T_i \rangle_{\mathrm{i}} \qquad \Longrightarrow \begin{bmatrix} T_0 \\ T_1 \end{bmatrix}$$

or to the summation expression as

$$\left\langle \sum_{\mathrm{i}} M_{ii} \right\rangle \qquad \Longrightarrow (M_{00} + M_{11} + M_{22})$$

EIN expressions

We use notation $\langle e \rangle_\alpha$ to represent an EIN expression e with tensor shape α. Subscripts can be bound to the outside of the EIN Expression as

$$\langle T_i \rangle_{\boxed{\mathrm{i}}} \qquad \Longrightarrow \begin{bmatrix} T_0 \\ T_1 \end{bmatrix}$$

or to the summation expression as

$$\left\langle \sum_{\boxed{\mathrm{i}}} M_{ii} \right\rangle \qquad \Longrightarrow (M_{00} + M_{11} + M_{22})$$

Mapping

Tensor operations on the surface language are mapped to concise EIN operators. The inner product between tensors is written in the Diderot syntax as

$$\textbf{tensor}[2]a;$$

$$\textbf{tensor}[2,3]b;$$

$$\textbf{tensor}[3]c = a \cdot b;$$

The operation is expressed with an EIN operator as

$$\text{c}{=}\lambda(T,M).\left\langle \sum_j T_j M_{ji} \right\rangle_i (a,b)$$

Family of Operators

EIN notation can represent a family of operators. The inner product is expressed with the generic EIN operator as

$$c=\lambda(T,R).\langle\textstyle\sum_i T_{\alpha i}R_{i\beta}\rangle_{\alpha\beta}(a,b)$$

Argument	Index(α,β)	expression
General	(α,β)	$\lambda(A,B)\langle\Sigma_k(A_{\alpha k}B_{k\beta})\rangle_{\alpha\beta}$
- vector ·vector	$(-,-)$	$\lambda(A,B)\langle\Sigma_k(A_k B_k)\rangle$
- matrix · vector	$(\alpha,-)$	$\lambda(A,B)\langle\Sigma_k(A_{\alpha k}B_k)\rangle_{\alpha}$
- vector · matrix	$(-,\beta)$	$\lambda(A,B)\langle\Sigma_k A_k B_{k\beta}\rangle_{\beta}$
- matrix ·matrix	(α,β)	$\lambda(A,B)\langle\Sigma_k(A_{\alpha k}B_{k\beta})\rangle_{\alpha\beta}$

Figure: Inner Product

Replaces direct-style operators "dot", "MulVecMat", "MulMatVec"...

EIN expressions

e	$::=$	T_α	**Tensor**	μ	$::=$	i, j, k	**Index Variables**
	$\mid$	F_α	**Field**		$\mid$	$\underline{c}$	**Index constant**
	$\mid$	$\mathcal{E}_{ijk}$	**Epsilon**				
	$\mid$	$\delta_{\mu,\mu}$	**Kronecker-deltas**	α	$::=$	$\overline{\mu}$	**Multi-index, list of indices**
	$\mid$	$\Sigma_\nu e$	**Summation**				
	$\mid$	$\frac{\partial}{\partial\alpha}$	**Derivative**	ν	$::=$	$[lb :: i :: ub]$	**Summation-range**
	$\mid$	$V_\alpha \circledast h^\alpha$	**Convolution**				
	$\mid$	$e(e)$	**Probe**	**E**	$::=$	$\lambda.\langle e\rangle_\alpha()$	**EIN Operator**
	$\mid$	$\widetilde{i}$	**Value-of-index**				
	$\mid$	$V_\alpha[\bar{e}]$	**Index-Images**				
	$\mid$	$\overline{h^{(\underline{c},i)}}[e]$	**Kernel**				
	$\mid$	$e + e$	**Addition**				
	$\mid$	ee	**Multiplication**				
	$\mid$	.....					

EIN Operator concisely represent tensor computations

e	$::=$	T_α	**Tensor**	μ	$::=$	i, j, k	**Index Variables**
	$\mid$	F_α	**Field**		$\mid$	$\underline{c}$	**Index constant**
	$\mid$	$\mathcal{E}_{ijk}$	**Epsilon**				
	$\mid$	$\delta_{\mu,\mu}$	**Kronecker-deltas**	α	$::=$	$\overline{\mu}$	**Multi-index, list of indices**
	$\mid$	$\Sigma_\nu e$	**Summation**				
	$\mid$	$\frac{\partial}{\partial \alpha}$	**Derivative**	ν	$::=$	$[lb :: i :: ub]$	**Summation-range**
	$\mid$	$V_\alpha \circledast h^\alpha$	**Convolution**				
	$\mid$	$e(e)$	**Probe**	**E**	$::=$	$\lambda.\langle e \rangle_\alpha()$	**EIN Operator**
	$\mid$	$\widetilde{i}$	**Value-of-index**				
	$\mid$	$\underline{V_\alpha[\bar{e}]}$	**Index-Images**				
	$\mid$	$h^{(\underline{c},i)}[e]$	**Kernel**				
	$\mid$	$e + e$	**Addition**				
	$\mid$	ee	**Multiplication**				
	$\mid$	.....					

Tensor Operators ($*, \cdot, :, \odot, \otimes, trace, transpose$)

Introduce Image Analysis EIN expressions

$$
\begin{array}{llll}
e & ::= & T_\alpha & \textbf{Tensor} \\
 & | & F_\alpha & \textbf{Field} \\
 & | & \mathcal{E}_{ijk} & \textbf{Epsilon} \\
 & | & \delta_{\mu,\mu} & \textbf{Kronecker-deltas} \\
 & | & \Sigma_\nu e & \textbf{Summation} \\
 & | & \frac{\partial}{\partial\alpha} & \textbf{Derivative} \\
 & | & V_\alpha \circledast h^\alpha & \textbf{Convolution} \\
 & | & \mathrm{e(e)} & \textbf{Probe} \\
 & | & \widetilde{i} & \textbf{Value-of-index} \\
 & | & V_\alpha[\bar{e}] & \textbf{Index-Images} \\
 & | & h^{\overline{(\underline{c},i)}}[e] & \textbf{Kernel} \\
 & | & e + e & \textbf{Addition} \\
 & | & ee & \textbf{Multiplication} \\
 & | & \ldots\ldots &
\end{array}
\qquad
\begin{array}{llll}
\mu & ::= & i, j, k & \textbf{Index Variables} \\
 & | & \underline{c} & \textbf{Index constant} \\
\alpha & ::= & \overline{\mu} & \textbf{Multi-index, list of indices} \\
\nu & ::= & [lb :: i :: ub] & \textbf{Summation-range} \\
\mathbf{E} & ::= & \lambda.\langle e \rangle_\alpha() & \textbf{EIN Operator}
\end{array}
$$

Field operators ($@$, $\circledast$) and reconstruction

Tensor Calcalus

e	$::=$	T_α	**Tensor**	μ	$::=$	i, j, k	**Index Variables**
	$\mid$	F_α	**Field**		$\mid$	$\underline{c}$	**Index constant**
	$\mid$	$\mathcal{E}_{ijk}$	**Epsilon**				
	$\mid$	$\delta_{\mu,\mu}$	**Kronecker-deltas**	α	$::=$	$\overline{\mu}$	**Multi-index, list of indices**
	$\mid$	$\Sigma_\nu e$	**Summation**				
	$\mid$	$\frac{\partial}{\partial\alpha}$	**Derivative**	ν	$::=$	$[lb :: i :: ub]$	**Summation-range**
	$\mid$	$V_\alpha \circledast h^\alpha$	**Convolution**				
	$\mid$	$e(e)$	**Probe**	$\mathbf{E}$	$::=$	$\lambda.\langle e \rangle_\alpha()$	**EIN Operator**
	$\mid$	$\widetilde{i}$	**Value-of-index**				
	$\mid$	$\underline{V_\alpha[\bar{e}]}$	**Index-Images**				
	$\mid$	$h^{(\underline{c},i)}[e]$	**Kernel**				
	$\mid$	$e + e$	**Addition**				
	$\mid$	ee	**Multiplication**				
	$\mid$	.....					

Operations $(\times, \nabla\times, identity)$

Field Operators

The **Gradient** is written in Diderot syntax as

$$\textbf{field } \#k(d)[\,]G = \nabla H$$

Then as an EIN operator

$$G = \lambda F.\left\langle \frac{\partial}{\partial x_i} F \right\rangle_i (H)$$

Differentiation operators $\nabla, \nabla\cdot, \nabla\times, \nabla\otimes$ are mapped to an EIN expressions that uses a handful of EIN operators $\frac{\partial}{\partial x_\alpha}, F_\beta, \Sigma$....
Then the EIN operators are optimized and expanded in a general fashion.

Field Operators

The **Gradient** is written in Diderot syntax as

$$\textbf{field}\ \#k(d)[\,]G = \nabla H$$

Then as an EIN operator

$$G = \lambda F.\left\langle \frac{\partial}{\partial x_i} F \right\rangle_i (H)$$

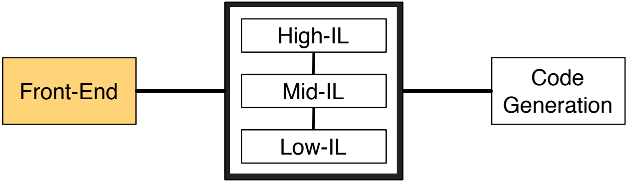
Differentiation operators $\nabla, \nabla\cdot, \nabla\times, \nabla\otimes$ are mapped to an EIN expressions that uses a handful of EIN operators $\frac{\partial}{\partial x_\alpha}, F_\beta, \Sigma \ldots.$
Then the EIN operators are optimized and expanded in a general fashion.

Field Operators

Description	Surface Language	EIN
Gradient	$\nabla\varphi$	$\lambda F.\left\langle \frac{\partial}{\partial x_i}\varphi \right\rangle_i ()$
Divergence	$\nabla \cdot F$	$\lambda F.\left\langle \Sigma_i \frac{\partial}{\partial x_i} F_i \right\rangle ()$
2d-Curl	$\nabla \times F$	$\lambda F.\left\langle \frac{\partial}{\partial x_{\underline{0}}} F_{\underline{1}} - \frac{\partial}{\partial x_{\underline{1}}} F_{\underline{0}} \right\rangle ()$
3d-Curl	$\nabla \times F$	$\lambda F.\left\langle \Sigma_{jk} \mathcal{E}_{ijk} \frac{\partial}{\partial x_j} F_k \right\rangle_i ()$
Laplacian	$\nabla \cdot \nabla\varphi$	$\lambda F.\left\langle \Sigma_i \frac{\partial}{\partial x_i^2}\varphi \right\rangle ()$
Jacobian	$\nabla \otimes F$	$\lambda F.\left\langle \frac{\partial}{\partial x_i} F_\alpha \right\rangle_{\alpha i} ()$
Hessian of Scalar Field	$\nabla \otimes \nabla F$	$\lambda F.\left\langle \frac{\partial}{\partial x_{ij}}\varphi \right\rangle_{ij} ()$
Hessian of Vector Field	$\nabla \otimes \nabla \otimes F$	$\lambda F.\left\langle \frac{\partial}{\partial x_{ij}} F_\alpha \right\rangle_{\alpha ij} ()$

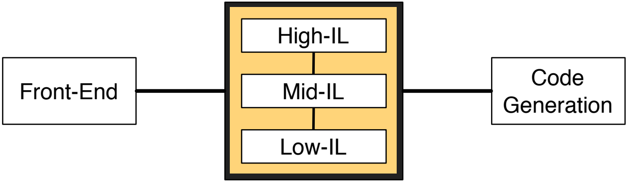
Implementation Overview

- All Tensor and Field Operators are mapped to EIN operators
- Optimizing and Lowering Phase
  - Index-based Optimizations and Simplifications
  - Simple-EIN, vectorization, and common subexpressions
  - Probed Fields are expanded
  - EIN Operators are transformed to scalar and vectors



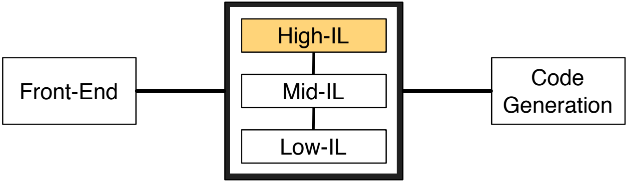
Implementation Overview

- All Tensor and Field Operators are mapped to EIN operators
- Optimizing and Lowering Phase
  - Index-based Optimizations and Simplifications
  - Simple-EIN, vectorization, and common subexpressions
  - Probed Fields are expanded
  - EIN Operators are transformed to scalar and vectors



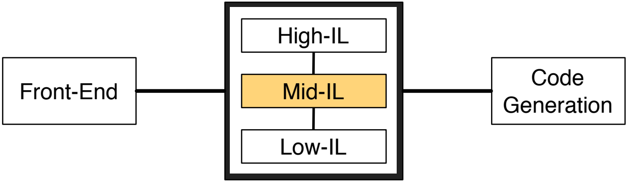
Implementation Overview

- All Tensor and Field Operators are mapped to EIN operators
- Optimizing and Lowering Phase
  - Index-based Optimizations and Simplifications
  - Simple-EIN, vectorization, and common subexpressions
  - Probed Fields are expanded
  - EIN Operators are transformed to scalar and vectors



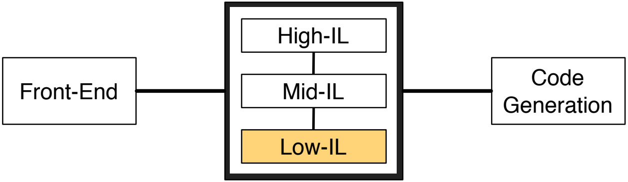
Implementation Overview

- All Tensor and Field Operators are mapped to EIN operators
- Optimizing and Lowering Phase
  - Index-based Optimizations and Simplifications
  - Simple-EIN, vectorization, and common subexpressions
  - Probed Fields are expanded
  - EIN Operators are transformed to scalar and vectors



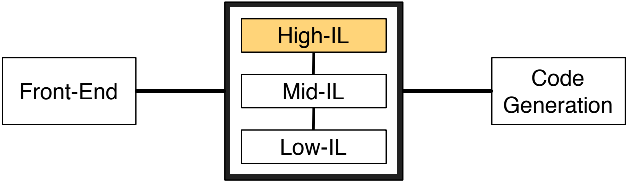
Implementation Overview

- All Tensor and Field Operators are mapped to EIN operators
- Optimizing and Lowering Phase
  - Index-based Optimizations and Simplifications
  - Simple-EIN, vectorization, and common subexpressions
  - Probed Fields are expanded
  - EIN Operators are transformed to scalar and vectors



Index-based Optimizations and Simplifications

- Systematic way of applying EIN operators that are mathematically well-founded
- Allow us to look at the structure of the larger operators and do index-based optimizations
- Term rewriting supported by tensor calculus



Index-based Substitutions

The simple act of substitution can also allow us to take advantage of index-base optimizations. In High-IL, the trace(a $\otimes$ b)

$$\begin{aligned} t_1 &= \lambda(T,R).\langle T_i R_j \rangle_{ij}(a,b) \\ t_2 &= \lambda M.\langle \Sigma_i M_{ii} \rangle (e_1) \end{aligned}$$

When we substitute the body of t_1 into t_2 the result will be (a $\cdot$ b).

$$\begin{aligned} t_3 &= \lambda(T,R).\langle \Sigma_i T_i R_i \rangle((a,b)) \\ \text{where } i \Longrightarrow i & \qquad j \Longrightarrow i \end{aligned}$$

Replaces rule Trace(outer(a,b))$\Longrightarrow$ dot(a,b).

Index-based Substitutions

Many identities fall out from this technique

Surface Language	Transition
Transpose(Transpose(T))	$\Longrightarrow T$
$\nabla \cdot \nabla\varphi$	$\Longrightarrow \nabla^2\varphi$
Trace($V(x)\otimes G(x)$)	$\Longrightarrow V(x) \cdot G(x)$
Trace($\nabla \otimes \nabla\varphi$)	$\Longrightarrow \nabla^2\varphi$
Trace(T+S)	$\Longrightarrow$trace(T)+ trace(S)
Trace ($a\otimes b$)	$\Longrightarrow a \cdot b$

Index-Based Term Rewriting

- The design allow us to look at the structure of subexpressions
- A handleful of rules allows us to find algebraic identities and do simplifications
- That otherwise would have required explicit rules in direct-style notation

$$\mathcal{E}_{ijk}\mathcal{E}_{ilm} \Longrightarrow \delta_{jl}\delta_{km} - \delta_{jm}\delta_{kl}$$

$$\delta_{ij}T_j \Longrightarrow T_i \qquad \mathcal{E}_{ijk}\frac{\partial}{\partial x_{jk}} \Longrightarrow 0$$

Surface Language	Transition
$\nabla \times \nabla\varphi$	$\Longrightarrow 0$
$\nabla\cdot(\nabla\times F)$	$\Longrightarrow 0$
$\nabla\times(\nabla\times A)$	$\Longrightarrow \nabla(\nabla \cdot A) - \nabla^2 A$
$(a\times b)\times c$	$\Longrightarrow b(a \cdot c) - a(b \cdot c)$
$a\times(b\times c)$	$\Longrightarrow b(a \cdot c) - c(a \cdot b)$
$(a\times b)\times(c\times d)$	$\Longrightarrow (a \cdot (c \times d))b - (b \cdot (c \times d))a$
$(a\times b)\cdot(c\times d)$	$\Longrightarrow (a \cdot c)(b \cdot d) - (a \cdot d)(b \cdot c)$

Field-Example

Consider the expression $\nabla \cdot \nabla s F(p)$

$$
\begin{array}{ll}
\textbf{field } k(d)[]F = V \circledast h; & F = \lambda(v,h).\langle v \circledast h\rangle(V,h) \\
\textbf{field } k(d)[]H = s * F; & H = \lambda(F,s).\langle s * F\rangle(F,s) \\
\textbf{field } k(d)[v]G = \nabla H; & G = \lambda\varphi.\left\langle \frac{\partial}{\partial x_i}\varphi \right\rangle_i (H) \\
\textbf{field } k(d)[]D = \nabla \cdot G; & D = \lambda F.\left\langle \Sigma_i \frac{\partial}{\partial x_i} F_i \right\rangle (G) \\
\textbf{tensor } []t = D(p); & t = \lambda(F,p).\langle D(p)\rangle(D,p)
\end{array}
$$

Field-Example

Consider the expression $\nabla \cdot \nabla sF(p)$

$$\begin{array}{ll}
\textbf{field } k(d)[]F = V \circledast h; & F = \lambda(v,h).\langle v \circledast h\rangle(V,h) \\
\textbf{field } k(d)[]H = s{*}F; & H = \lambda(F,s).\langle s * F\rangle(F,s) \\
\textbf{field } k(d)[v]G = \nabla H; & \mathrm{G}{=}\lambda\varphi.\left\langle \frac{\partial}{\partial x_i}\varphi \right\rangle_i (H) \\
\textbf{field } k(d)[]D = \nabla{\cdot}G; & \mathrm{D}{=}\lambda F.\left\langle \Sigma_i \frac{\partial}{\partial x_i} F_i \right\rangle (G) \\
\textbf{tensor } []t = D(p); & t = \lambda(F,p).\langle D(p)\rangle(D,p)
\end{array}$$

Field-Example-Laplacian

Combinations of EIN operators are used to represent more complicated expressions in the surface language. Earlier we saw expression $(\nabla \cdot \nabla H)$

$$\begin{aligned} \textbf{Gradient} &= \lambda\varphi.\left\langle \frac{\partial}{\partial x_j}\varphi \right\rangle_j (H) \\ \textbf{Divergence} &= \lambda F.\left\langle \sum_i \frac{\partial}{\partial x_i} \cdot F_i \right\rangle (\textbf{Gradient}) \end{aligned}$$

The new expression will be the Laplacian of the scalar field.

$$\textbf{Laplacian} = \begin{array}{l} \lambda\varphi.\left\langle \frac{\partial}{\partial x_i x_i}\varphi \right\rangle (H) \\ \text{where } j \Longrightarrow i \end{array}$$

Field-Example-Normalization

Various rewrites that involve making operations on the tensor result rather than the the entire field. Consider the expression $\nabla \cdot \nabla sF(p)$

$$t = \lambda(v,h,p,s).\left\langle \Sigma_i \frac{\partial}{\partial x_i}\frac{\partial}{\partial x_i}(s * v \circledast h(p)) \right\rangle (v,h,p)$$

Combine differentiation expression

$$\Longrightarrow \left\langle \Sigma_i \frac{\partial}{\partial x_{ii}}(s * v \circledast h(p)) \right\rangle$$

Move constants outside differentiation and probing.

$$\Longrightarrow \left\langle s * \Sigma_i \frac{\partial}{\partial x_{ii}}(v \circledast h(p)) \right\rangle$$

Push differentiation index to kernel

$$\Longrightarrow \left\langle s * \Sigma_i (v \circledast h^{ii}(p)) \right\rangle$$

Field-Example-Field Expansion

The field expression we saw earlier $\nabla \cdot \nabla s F(p)$ is written as single EIN operator in High-IL as

$$\mathrm{t} = \lambda(s, v, h, p).\left\langle s * \Sigma_i v \circledast h^{ii}(p)\right\rangle(s, v, h, p)$$

Ein Operators are split into simple-EIN operators.

$$\mathrm{e}{=}\left\langle \Sigma_i v \circledast h^{ii}(p)\right\rangle$$

$$\mathrm{c}{=}\lambda(v, h, p).\langle e\rangle(v, h, p) \qquad \mathrm{t}{=}\lambda(s, c).\langle s * c\rangle(s, c)$$

Then the probed field is expressed as

$$\mathrm{e} \Longrightarrow \left\langle \Sigma_{jk}(V[n_{\underline{0}} + \widetilde{j}, n_{\underline{1}} + \widetilde{k}])(h^{\delta_{0i}+\delta_{0i}}[f_{\underline{0}} - \widetilde{j}])(h^{\delta_{1i}+\delta_{1i}}[f_{\underline{1}} - \widetilde{k}])\right\rangle$$

Field-Example-Field Expansion

The field expression we saw earlier $\nabla \cdot \nabla s F(p)$ is written as single EIN operator in High-IL as

$$\text{t} = \lambda(s, v, h, p).\Big\langle s * \Sigma_i v \circledast h^{ii}(p) \Big\rangle (s, v, h, p)$$

Ein Operators are split into simple-EIN operators.

$$\text{e} = \Big\langle \Sigma_i v \circledast h^{ii}(p) \Big\rangle$$

$$\text{c} = \lambda(v, h, p).\langle e \rangle (v, h, p) \qquad \text{t} = \lambda(s, c).\langle s * c \rangle (s, c)$$

Then the probed field is expressed as

$$\text{e} \Longrightarrow \Big\langle \Sigma_{jk} (V[n_{\underline{0}} + \widetilde{j}, n_{\underline{1}} + \widetilde{k}])(h^{\delta_{0i} + \delta_{0i}}[f_{\underline{0}} - \widetilde{j}])(h^{\delta_{1i} + \delta_{1i}}[f_{\underline{1}} - \widetilde{k}]) \Big\rangle$$

Field-Example in Low-IL

Mid-IL

$$c = \Sigma_{jk}(V[n_{\underline{0}} + \widetilde{j}, n_{\underline{1}} + \widetilde{k}])(h^{\delta_{0i}+\delta_{0i}}[f_{\underline{0}} - \widetilde{j}])(h^{\delta_{1i}+\delta_{1i}}[f_{\underline{1}} - \widetilde{k}])$$

In Low-IL the summation is unrolled as

$$\Longrightarrow \Sigma_{jk}(V[n_{\underline{0}} + \widetilde{j}, n_{\underline{1}} + \widetilde{k}])(h^{\delta_{00}+\delta_{00}}[f_{\underline{0}} - \widetilde{j}])(h^{\delta_{10}+\delta_{10}}[f_{\underline{1}} - \widetilde{k}])$$

$$+\Sigma_{jk}(V[n_{\underline{0}} + \widetilde{j}, n_{\underline{1}} + \widetilde{k}])(h^{\delta_{01}+\delta_{01}}[f_{\underline{0}} - \widetilde{j}])(h^{\delta_{11}+\delta_{11}}[f_{\underline{1}} - \widetilde{k}])$$

Deltas are evaluated to integers

$$\Longrightarrow \Sigma_{jk}(V[n_{\underline{0}} + \widetilde{j}, n_{\underline{1}} + \widetilde{k}])(h^{2}[f_{\underline{0}} - \widetilde{j}])(h^{0}[f_{\underline{1}} - \widetilde{k}])$$

$$+\Sigma_{jk}(V[n_{\underline{0}} + \widetilde{j}, n_{\underline{1}} + \widetilde{k}])(h^{0}[f_{\underline{0}} - \widetilde{j}])(h^{2}[f_{\underline{1}} - \widetilde{k}])$$

Then the compiler expands out these evaluations.

Field-Example in Low-IL

Mid-IL

$$c = \Sigma_{jk}(V[n_{\underline{0}} + \widetilde{j}, n_{\underline{1}} + \widetilde{k}])(h^{\delta_{0i}+\delta_{0i}}[f_{\underline{0}} - \widetilde{j}])(h^{\delta_{1i}+\delta_{1i}}[f_{\underline{1}} - \widetilde{k}])$$

In Low-IL the summation is unrolled as

$$\Longrightarrow \Sigma_{jk}(V[n_{\underline{0}} + \widetilde{j}, n_{\underline{1}} + \widetilde{k}])(h^{\delta_{00}+\delta_{00}}[f_{\underline{0}} - \widetilde{j}])(h^{\delta_{10}+\delta_{10}}[f_{\underline{1}} - \widetilde{k}])$$

$$+\Sigma_{jk}(V[n_{\underline{0}} + \widetilde{j}, n_{\underline{1}} + \widetilde{k}])(h^{\delta_{01}+\delta_{01}}[f_{\underline{0}} - \widetilde{j}])(h^{\delta_{11}+\delta_{11}}[f_{\underline{1}} - \widetilde{k}])$$

Deltas are evaluated to integers

$$\Longrightarrow \Sigma_{jk}(V[n_{\underline{0}} + \widetilde{j}, n_{\underline{1}} + \widetilde{k}])(h^{2}[f_{\underline{0}} - \widetilde{j}])(h^{0}[f_{\underline{1}} - \widetilde{k}])$$

$$+\Sigma_{jk}(V[n_{\underline{0}} + \widetilde{j}, n_{\underline{1}} + \widetilde{k}])(h^{0}[f_{\underline{0}} - \widetilde{j}])(h^{2}[f_{\underline{1}} - \widetilde{k}])$$

Then the compiler expands out these evaluations.

Field-Example in Low-IL

Mid-IL

$$c = \Sigma_{jk}(V[n_{\underline{0}} + \widetilde{j}, n_{\underline{1}} + \widetilde{k}])(h^{\delta_{0i}+\delta_{0i}}[f_{\underline{0}} - \widetilde{j}])(h^{\delta_{1i}+\delta_{1i}}[f_{\underline{1}} - \widetilde{k}])$$

In Low-IL the summation is unrolled as

$$\Longrightarrow \Sigma_{jk}(V[n_{\underline{0}} + \widetilde{j}, n_{\underline{1}} + \widetilde{k}])(h^{\delta_{00}+\delta_{00}}[f_{\underline{0}} - \widetilde{j}])(h^{\delta_{10}+\delta_{10}}[f_{\underline{1}} - \widetilde{k}])$$

$$+\Sigma_{jk}(V[n_{\underline{0}} + \widetilde{j}, n_{\underline{1}} + \widetilde{k}])(h^{\delta_{01}+\delta_{01}}[f_{\underline{0}} - \widetilde{j}])(h^{\delta_{11}+\delta_{11}}[f_{\underline{1}} - \widetilde{k}])$$

Deltas are evaluated to integers

$$\Longrightarrow \Sigma_{jk}(V[n_{\underline{0}} + \widetilde{j}, n_{\underline{1}} + \widetilde{k}])(h^{2}[f_{\underline{0}} - \widetilde{j}])(h^{0}[f_{\underline{1}} - \widetilde{k}])$$

$$+\Sigma_{jk}(V[n_{\underline{0}} + \widetilde{j}, n_{\underline{1}} + \widetilde{k}])(h^{0}[f_{\underline{0}} - \widetilde{j}])(h^{2}[f_{\underline{1}} - \widetilde{k}])$$

Then the compiler expands out these evaluations.

Related Work

- **Spiral** is a DSL that supports various digitial signal processing. They use signal processing language (SPL) that includes a small set of constructs such as matrix operators, and krnocker product.
- **TCE** is a Tensor contraction Engine created to represent quantum chemistry. Creates algorthms for CSE in large search space, and decides how to cost-effectively mutiply tensors by using libraries.
- **Ahlander** provides support for index notation on the surface language and implements as a C++ library.
- **Index Notation** ambiguities have been represented differently to support a diverse set of operations.

Future Work

- More Tensor Calculus ($F \cdot G$, $F \times G$,..), and identities
- Multiplying Tensors leaves many stones unturned.
- Find common subexpressions in large complicated EIN operators
- Code Generation Implementation

Conclusion

- We presented the design and implementation of EIN Operators
- Advantages included in this representation
- Extend operators suported on surface lanuage
- Plan to lift a richer set of tensor operators up to fields

Bibliography

- T. Henretty S. Krishnamoorthy H. Zhang G. Baumgartner D. E. Bernholdt M. Nooijen R. M. Pitzer J. Ramanujam A. Hartono, Q. Lu and P. Sadayappan. Performance optimization of tensor contraction expressions for many-body methods in quantum chemistry. The Journal of Physical Chemistry, 113(45):1271512723, 2009.
- Wenjing Ma Sriram Krishnamoorthy Oreste Villa Karol Kowalski Gagan Agrawal. Optimizing tensor contraction expressions for hybrid cpu-gpu execution. Cluster Computing Special Issue, 2011.
- K. Ahlander. Einstein summation for multi-dimensional arrays. An International Journal computers and mathematics with applications, pages 10071017, December 2002.
- Marcel Nooijen Gerald Baumgartner David E. B Albert Hartono, Alexander Sibiryakov. Automated operation minimization of tensor contraction expressions in electronic structure calculations. Proc. ICCS 2005 5th International Conference on Computational Science, pages 155164, 2005.
- A. Barr. The einstein summation notation, introduction to cartesian ten- sors and extensions to the notation. Draft paper; available at url- http://zeus.phys.uconn.edu/mcintyre/workfiles/Papers/Einstien-Summation-Notation.pdf.
- E. Bolton. A simple notation for differential vector expressions in orthogonal curvilinear coordinates. Geo-physical Journal International, 115(3):654666, December 1999.
- Brian Cabral and Leith Casey Leedom. Imaging vector fields using line integral convolution. pages 263270, August 1993.
- Charisee Chiw, Gordon Kindlmann, John Reppy, Lamont Samuels, and Nick Seltzer. Diderot: A parallel DSL for image analysis and visualization. pages 111120, 2012.
- Tai L. Chow. Mathematical Methods for Physicists : A Concise Introduction. CAMBRIDGE UNIVERSITY PRESS, Cambridge, 2000.
- Ron Cytron, Jeanne Ferrante, Barry K. Rosen, Mark N. Wegman, and F. Kenneth Zadeck. Efficiently computing static single assignment form and the control dependence graph. ACM Transactions on Programming Languages and Systems, 13(4):451490, Oct 1991.
- Kees Dullemond and Kasper Peeters. Introduction to Tensor Calculus.
- Voronenko Y. Pu schel M. Franchetti, F. A rewriting system for the vectorization of signal transforms. In: High Performance Computing for Computational Science (VECPAR), 4395, 2006. 52

Questions?

Tensor Operations

Consider the inner product between vector T and a 2-by-3 matrix (M).

$$C = T \cdot M$$

$$C_i = \langle \Sigma_j T_j M_{ji} \rangle_i$$

$$\textbf{where } [0 : i : 2], \text{ and } [0 : j : 1]$$

Iterate over the outer index i to get the structure as the tensor result

$$C_{ij} = \begin{bmatrix} \Sigma_j T_j M_{j0} \\ \Sigma_j T_j M_{j1} \\ \Sigma_j T_j M_{j2} \end{bmatrix}$$

Unroll the summation in the expression as

$$C_{ij} = \begin{bmatrix} T_0 M_{00} & T_1 M_{10} \\ T_0 M_{01} & T_1 M_{11} \\ T_0 M_{02} & T_1 M_{12} \end{bmatrix}$$

Family of Operators

EIN notation can represent a family of Operators. The inner product between tensors can be represented in the Diderot Syntax as

$$\textbf{tensor}[\sigma]a;$$

$$\textbf{tensor}[\sigma']b;$$

$$\textbf{tensor}[\varsigma]c = a{\cdot}b;$$

The inner product is expressed with the generic EIN operator as

$$\text{c}{=}\lambda(T,R).\langle\textstyle\sum_i T_{\alpha i}R_{i\beta}\rangle_{\alpha\beta}(a,b)$$

$$\textbf{where } \sigma = \alpha :: i, \sigma' = i :: \beta, \text{ and } \varsigma = \alpha\beta$$

Replaces direct-style operators “dot-product”,“vector-matrix”...

High-IL Rewrites Tensor Example

$e = (a \times (b \times c))$
The cross product is transformed to two operators as
$d = \lambda(A,B)\langle \Sigma_{yz}\mathcal{E}_{xyz}A_yB_z\rangle_x(b,c)$ $\quad e = \lambda(A,B)\langle \Sigma_{jk}\mathcal{E}_{ijk}A_jB_k\rangle_i(a,d)$
The expression is rewritten as

$$
\begin{array}{llc}
\Longrightarrow & \lambda(a,b,c) & \langle \Sigma_{jklm}\mathcal{E}_{ijk}\mathcal{E}_{klm}a_jb_lc_m\rangle_i(a,b,c) \\
\Longrightarrow & \lambda(a,b,c) & \langle \Sigma_{jlm}(\delta_{il}\delta_{jm}a_jb_lc_m) - \Sigma_{jlm}(\delta_{im}\ \delta_{jk}a_jb_lc_m)\rangle_i \\
\Longrightarrow & \lambda(a,b,c) & \langle \Sigma_j(a_jb_ic_j) - \Sigma_j(a_jb_jc_i)\rangle_i \\
\Longrightarrow & & b(a\cdot c) - a(b\cdot c)
\end{array}
$$

Simple-EIN

- Simple-EIN are EIN operators with just one computation
- Challenges to EIN operators
  - EIN Operators can get large and complicated
  - Determine order of operations for tensor multiplication
  - Find vector operations for code generation
  - Large search space makes it harder to find common subexpressions that could produce redundant code
- EIN operators are transformed to Simple-EIN operators

Simple-Ein Vectorization Potential

Consider expression $a{+}(b{\cdot}c)$

$$t = \lambda(A,B,C).\left\langle A_i + (\sum_j B_{ij}C_j) \right\rangle_i (a,b,c)$$

EIN operator t is split into two EIN operators e_1 and e_2, each with a single operation. These mid-IL EIN operators are trivially rewritten as low-IL vector product and vector addition operators as

$$e_1 = \lambda(A,B).\left\langle \sum_j A_{ij}B_j \right\rangle_i (b,c)$$

$$e_2 = \lambda(A,B).\langle A_i + B_i \rangle_i (A, e_1)$$

Simple-Ein-Vectorization Potential

Simple Ein can be used to show vectorization potential and find redundant expressions. In the expression $a+(b \cdot c)$

$$t = \lambda(A,B,C).\left\langle A_i + (\sum_j B_{ij} C_j) \right\rangle_i (a,b,c)$$

In this case, the EIN operator t is split into two EIN operators e_1 and e_2, each with a single operation. These mid-IL EIN operators are trivially rewritten as low-IL vector product and vector addition operators as

$$e_1 = \lambda(A,B).\left\langle \sum_j A_{ij} B_j \right\rangle_i (b,c)$$

$$e_2 = \lambda(A,B).\langle A_i + B_i \rangle_i (A, e_1)$$

If we had (c · transpose(b))

$$e_1' = \lambda(A,B).\left\langle \sum_j B_j A_{ji} \right\rangle_i (b,c)$$

We would need to consider using different techniques to easily find vectorization potential.

Mid-IL Field Expansion

In High-IL the Probing of a field at a position is generally expressed as

$$\left\langle v_\alpha \circledast h^\beta(x) \right\rangle_{\alpha\beta}$$

Transform world-space position to image-space.

$$n = [M^{-1}x] \text{ and } f = M^{-1}x - n$$

The probing of a 2-d Field is expressed in Mid-IL as

$$\Longrightarrow \left\langle \Sigma_{jk}(v_\alpha[n_{\underline{0}} + \widetilde{j}, n_{\underline{1}} + \widetilde{k}])(h^{\delta_{0\beta}\cdot\cdot}[f_{\underline{0}} - \widetilde{j}])(h^{\delta_{1\beta}\cdot\cdot}[f_{\underline{1}} - \widetilde{k}]) \right\rangle_{\alpha\beta}$$

Generalization is important because we do not need a case by case analysis to represent fields.

EIN Operators are transformed to scalar and vectors

- Map mid-IL simple-EIN operators to low-IL operators
- Iterate over the shape of tensor expression then evaluate subexpressions

$$C = \lambda(A, B).\left\langle \sum_j A_{ij} B_j \right\rangle_i (b, c)$$

$$C = \begin{bmatrix} \sum_j A_{0j} B_j \\ \sum_j A_{1j} B_j \end{bmatrix}$$

- Low-IL Operators designed to take advantage of target hardware
- Intend to map vector operators to finite-size operations

Field-Example

Consider the expression $\nabla \cdot \nabla s F(p)$

Field	**field** $k(d)[]F = V \circledast h;$	$F = \lambda(v,h).\langle v \circledast h\rangle(V,h)$
Scaling	**field** $k(d)[]H = s{*}F;$	$H = \lambda(F,s).\langle s * F\rangle(F,s)$
Gradient	**field** $k(d)[v]G = \nabla H;$	$G = \lambda\varphi.\left\langle \frac{\partial}{\partial x_i}\varphi \right\rangle_i(H)$
Divergence	**field** $k(d)[]D = \nabla{\cdot}G;$	$D = \lambda F.\left\langle \Sigma_i \frac{\partial}{\partial x_i} F_i \right\rangle(G)$
Probed Field	**tensor** $[]t = D(p);$	$t = \lambda(F,p).\langle D(p)\rangle(D,p)$