resophonic::kamasu

Computing on the GPU with CUDA and boost::proto

Troy D. Straszheim

Resophonic Systems, Inc.

Washington, DC

The Underpants Gnomes' Business Plan

1. Collect Underpants
2. ???
3. Profit

The Kamasu Business Plan

1. Learn boost::proto
2. Learn CUDA
3. Combine
4. ???
5. Profit

The Kamasu Business Plan

1. Learn boost::proto
2. Learn CUDA
3. Combine
4. ??? **<- You are here**
5. Profit

Outline

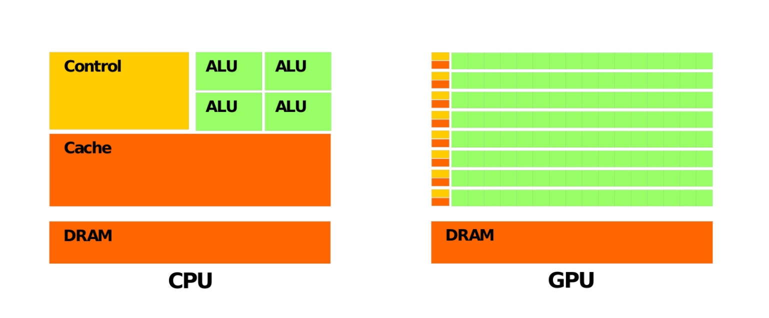
CUDA

boost::proto
- transforms

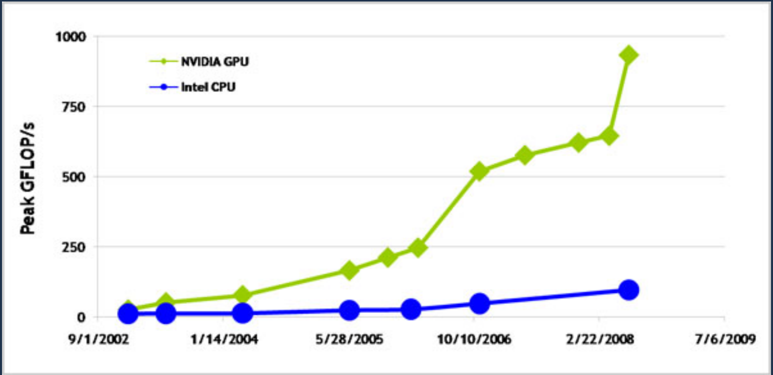
Related efforts

resophonic::kamasu
- benchmarks

The GPU vs the CPU

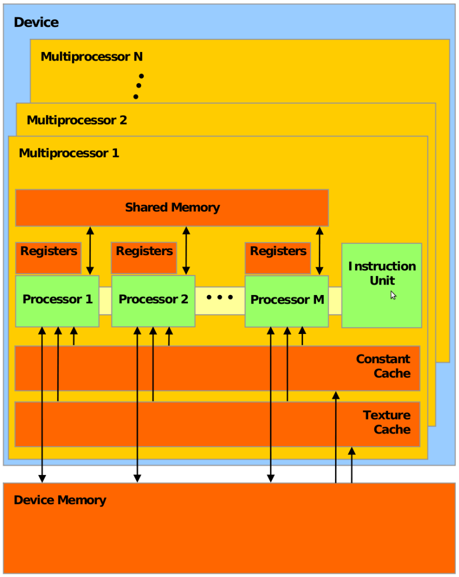


The GPU vs the CPU



(image courtesy of NVIDIA)

The GPU


Device
Multiprocessor N
Multiprocessor 2
Multiprocessor 1
Shared Memory
Registers
Registers
Registers
Processor 1
Processor 2
Processor M
Instruction Unit
Constant Cache
Texture Cache
Device Memory

Building software to run on the gpu

- “C for CUDA” is (soon C++) with extensions
- Mix device and host code (and both code)
- Run the file through NVCC
- compile the result with gcc, compiled CUDA code is linked into the binary

Building software to run on the gpu

- “C for CUDA” is (soon C++) with extensions
- Mix device and host code (and both code)
- Run the file through NVCC
- compile the result with gcc, compiled CUDA code is linked into the binary

Building software to run on the gpu

- “C for CUDA” is (soon C++) with extensions
- Mix device and host code (and both code)
- Run the file through NVCC
- compile the result with gcc, compiled CUDA code is linked into the binary

Building software to run on the gpu

- “C for CUDA” is (soon C++) with extensions
- Mix device and host code (and both code)
- Run the file through NVCC
- compile the result with gcc, compiled CUDA code is linked into the binary

Serially adding a scalar to an array

```
void add(float* data, unsigned size,
         float scalar)
{
  for(unsigned i=0; i<size; i++)
    data[i] += scalar;
}
```

Adding a scalar to an array in cuda-parallel

```
__global__ void
add(float *data, float scalar)
{
  data[threadIdx.x] += scalar;
}
```

Adding a scalar to an array in cuda-parallel

```
__global__ void
add(float *data, float scalar)
{
  data[threadIdx.x] += scalar;
}
```

Adding a scalar to an array in cuda-parallel

```
__global__ void
add(float *data, float scalar)
{
  data[threadIdx.x] += scalar;
}
```

Adding a scalar to an array in cuda-parallel

```
__global__ void
add(float *data, float scalar)
{
  data[threadIdx.x] += scalar;
}

int main() {
  int N = 256;
  float *arr = make_vector_on_gpu(N);

  add<<<1, N>>>(arr, 3.14159);
}
```

Adding a scalar to an array in cuda-parallel

```
__global__ void
add(float *data, float scalar)
{
  data[threadIdx.x] += scalar;
}

int main() {
  int N = 256;
  float *arr = make_vector_on_gpu(N);

  add<<<1, N>>>(arr, 3.14159);
}
```

Adding a scalar to an array in cuda-parallel

```
__global__ void
add(float *data, float scalar)
{
  data[threadIdx.x] += scalar;
}

int main() {
  int N = 256;
  float *arr = make_vector_on_gpu(N);

  add<<<1, N>>>(arr, 3.14159);
}
```

Adding a scalar to an array in cuda-parallel

```
__global__ void
add(float *data, float scalar)
{
  data[threadIdx.x] += scalar;
}

int main() {
  int N = 256;
  float *arr = make_vector_on_gpu(N);

  add<<<1, N>>>(arr, 3.14159);
}
```

Adding a scalar to an array in cuda-parallel

```
__global__ void
add(float *data, float scalar)
{
  data[threadIdx.x] += scalar;
}

int main() {
  int N = 256;
  float *arr = make_vector_on_gpu(N);

  add<<<1, N>>>(arr, 3.14159);
}
```

Adding a scalar to an array in cuda-parallel

```
__global__ void
add(float *data, float scalar)
{
  data[threadIdx.x] += scalar;
}

int main() {
  int N = 256;
  float *arr = make_vector_on_gpu(N);

  add<<<1, N>>>(arr, 3.14159);
}
```

Adding a scalar to an array in cuda-parallel

```
__global__ void
add(float *data, float scalar)
{
  data[threadIdx.x] += scalar;
}

int main() {
  int N = 256;
  float *arr = make_vector_on_gpu(N);

  add<<<1, N>>>(arr, 3.14159);
}
```

Adding a scalar to an array in cuda-parallel

```
__global__ void
add(float *data, float scalar)
{
  data[threadIdx.x] += scalar;
}

int main() {
  int N = 256;
  float *arr = make_vector_on_gpu(N);

  add<<<1, N>>>(arr, 3.14159);
}
```

Adding a scalar to an array in cuda-parallel

```
__global__ void
add(float *data, float scalar)
{
  data[threadIdx.x] += scalar;
}

int main() {
  int N = 256;
  float *arr = make_vector_on_gpu(N);

  add<<<1, N>>>(arr, 3.14159);
}
```

CUDA Memory management

```
cudaError_t cudaMalloc(void** devPtr, size_t count );
cudaError_t cudaFree(void* devPtr);

cudaError_t cudaMemcpy(void* dst, const void* src,
                       size_t count,
                       enum cudaMemcpyKind kind);

cudaMemcpyHostToHost
cudaMemcpyHostToDevice
cudaMemcpyDeviceToHost
cudaMemcpyDeviceToDevice

cudaError_t cudaMemset(void* devPtr, int value,
                       size_t count);
```

CUDA Memory management

```
cudaError_t cudaMalloc(void** devPtr, size_t count );
cudaError_t cudaFree(void* devPtr);

cudaError_t cudaMemcpy(void* dst, const void* src,
                       size_t count,
                       enum cudaMemcpyKind kind);

cudaMemcpyHostToHost
cudaMemcpyHostToDevice
cudaMemcpyDeviceToHost
cudaMemcpyDeviceToDevice

cudaError_t cudaMemset(void* devPtr, int value,
                       size_t count);
```

CUDA Memory management

```
cudaError_t cudaMalloc(void** devPtr, size_t count );
cudaError_t cudaFree(void* devPtr);

cudaError_t cudaMemcpy(void* dst, const void* src,
                       size_t count,
                       enum cudaMemcpyKind kind);

cudaMemcpyHostToHost
cudaMemcpyHostToDevice
cudaMemcpyDeviceToHost
cudaMemcpyDeviceToDevice

cudaError_t cudaMemset(void* devPtr, int value,
                       size_t count);
```

CUDA Memory management

```
cudaError_t cudaMalloc(void** devPtr, size_t count );
cudaError_t cudaFree(void* devPtr);

cudaError_t cudaMemcpy(void* dst, const void* src,
                       size_t count,
                       enum cudaMemcpyKind kind);

cudaMemcpyHostToHost
cudaMemcpyHostToDevice
cudaMemcpyDeviceToHost
cudaMemcpyDeviceToDevice

cudaError_t cudaMemset(void* devPtr, int value,
                       size_t count);
```

CUDA Memory management

```
cudaError_t cudaMalloc(void** devPtr, size_t count );
cudaError_t cudaFree(void* devPtr);

cudaError_t cudaMemcpy(void* dst, const void* src,
                       size_t count,
                       enum cudaMemcpyKind kind);

cudaMemcpyHostToHost
cudaMemcpyHostToDevice
cudaMemcpyDeviceToHost
cudaMemcpyDeviceToDevice

cudaError_t cudaMemset(void* devPtr, int value,
                       size_t count);
```

CUDA Memory management

```
cudaError_t cudaMalloc(void** devPtr, size_t count );
cudaError_t cudaFree(void* devPtr);

cudaError_t cudaMemcpy(void* dst, const void* src,
                       size_t count,
                       enum cudaMemcpyKind kind);

cudaMemcpyHostToHost
cudaMemcpyHostToDevice
cudaMemcpyDeviceToHost
cudaMemcpyDeviceToDevice

cudaError_t cudaMemset(void* devPtr, int value,
                       size_t count);
```

CUDA Memory management

```
cudaError_t cudaMalloc(void** devPtr, size_t count );
cudaError_t cudaFree(void* devPtr);

cudaError_t cudaMemcpy(void* dst, const void* src,
                       size_t count,
                       enum cudaMemcpyKind kind);

cudaMemcpyHostToHost
cudaMemcpyHostToDevice
cudaMemcpyDeviceToHost
cudaMemcpyDeviceToDevice

cudaError_t cudaMemset(void* devPtr, int value,
                       size_t count);
```

CUDA Memory management

```
cudaError_t cudaMalloc(void** devPtr, size_t count );
cudaError_t cudaFree(void* devPtr);

cudaError_t cudaMemcpy(void* dst, const void* src,
                       size_t count,
                       enum cudaMemcpyKind kind);

cudaMemcpyHostToHost
cudaMemcpyHostToDevice
cudaMemcpyDeviceToHost
cudaMemcpyDeviceToDevice

cudaError_t cudaMemset(void* devPtr, int value,
                       size_t count);
```

CUDA Memory management

```
cudaError_t cudaMalloc(void** devPtr, size_t count );
cudaError_t cudaFree(void* devPtr);

cudaError_t cudaMemcpy(void* dst, const void* src,
                       size_t count,
                       enum cudaMemcpyKind kind);

cudaMemcpyHostToHost
cudaMemcpyHostToDevice
cudaMemcpyDeviceToHost
cudaMemcpyDeviceToDevice

cudaError_t cudaMemset(void* devPtr, int value,
                       size_t count);
```

CUDA Memory management

```
cudaError_t cudaMalloc(void** devPtr, size_t count );
cudaError_t cudaFree(void* devPtr);

cudaError_t cudaMemcpy(void* dst, const void* src,
                       size_t count,
                       enum cudaMemcpyKind kind);

cudaMemcpyHostToHost
cudaMemcpyHostToDevice
cudaMemcpyDeviceToHost
cudaMemcpyDeviceToDevice

cudaError_t cudaMemset(void* devPtr, int value,
                       size_t count);
```

A holder class

```
template <typename T>
class holder : boost::noncopyable
{
    T* devmem;
    std::size_T size_;

  public:

    holder();
    holder(std::size_t n);
    ~holder();
    boost::shared_ptr<holder> clone();
    void resize(std::size_t size);
    T* data() { return devmem; }
    std::size_t size() { return size_; }
};
```

A holder class

```
template <typename T>
class holder : boost::noncopyable
{
    T* devmem;
    std::size_T size_;

  public:

    holder();
    holder(std::size_t n);
    ~holder();
    boost::shared_ptr<holder> clone();
    void resize(std::size_t size);
    T* data() { return devmem; }
    std::size_t size() { return size_; }
};
```

A holder class

```
template <typename T>
class holder : boost::noncopyable
{
    T* devmem;
    std::size_T size_;

  public:

    holder();
    holder(std::size_t n);
    ~holder();
    boost::shared_ptr<holder> clone();
    void resize(std::size_t size);
    T* data() { return devmem; }
    std::size_t size() { return size_; }
};
```

A holder class

```
template <typename T>
class holder : boost::noncopyable
{
    T* devmem;
    std::size_T size_;

  public:

    holder();
    holder(std::size_t n);
    ~holder();
    boost::shared_ptr<holder> clone();
    void resize(std::size_t size);
    T* data() { return devmem; }
    std::size_t size() { return size_; }
};
```

A holder class

```
template <typename T>
class holder : boost::noncopyable
{
    T* devmem;
    std::size_T size_;

  public:

    holder();
    holder(std::size_t n);
    ~holder();
    boost::shared_ptr<holder> clone();
    void resize(std::size_t size);
    T* data() { return devmem; }
    std::size_t size() { return size_; }
};
```

A holder class

```
template <typename T>
class holder : boost::noncopyable
{
    T* devmem;
    std::size_T size_;

  public:

    holder();
    holder(std::size_t n);
    ~holder();
    boost::shared_ptr<holder> clone();
    void resize(std::size_t size);
    T* data() { return devmem; }
    std::size_t size() { return size_; }
};
```

A holder class

```
template <typename T>
class holder : boost::noncopyable
{
    T* devmem;
    std::size_T size_;

  public:

    holder();
    holder(std::size_t n);
    ~holder();
    boost::shared_ptr<holder> clone();
    void resize(std::size_t size);
    T* data() { return devmem; }
    std::size_t size() { return size_; }
};
```

A holder class

```
template <typename T>
class holder : boost::noncopyable
{
    T* devmem;
    std::size_T size_;

  public:

    holder();
    holder(std::size_t n);
    ~holder();
    boost::shared_ptr<holder> clone();
    void resize(std::size_t size);
    T* data() { return devmem; }
    std::size_t size() { return size_; }
};
```

holder<T> implementations

```
template <typename T>
holder<T>::holder(std::size_t s) : size_(s)
{
   cudaMalloc(reinterpret_cast<void**>(&devmem),
              size_ * sizeof(T));
}

template <typename T>
holder<T>::~holder()
{
  if (devmem)
    cudaFree(devmem);
}
```

holder<T> implementations

```
template <typename T>
holder<T>::holder(std::size_t s) : size_(s)
{
   cudaMalloc(reinterpret_cast<void**>(&devmem),
              size_ * sizeof(T));
}

template <typename T>
holder<T>::~holder()
{
  if (devmem)
    cudaFree(devmem);
}
```

holder<T> implementations

```
template <typename T>
boost::shared_ptr<holder<T> >
holder<T>::clone()
{
  boost::shared_ptr<holder> nh(new holder(size_));
  cudaMemcpy(devmem, nh->devmem,
             sizeof(T) * size_,
             cudaMemcpyDeviceToDevice);
  return nh;
}
```

holder<T> implementations

```
template <typename T>
void
holder<T>::put(const T* hostmem, std::size_t s)
{
  if (s != size_)
    resize(s);
  cudaMemcpy(devmem, hostmem, s * sizeof(T),
             cudaMemcpyHostToDevice);
}

template <typename T>
void
holder<T>::get(T* hostmem)
{
  cudaMemcpy(hostmem, devmem, size_ * sizeof(T),
             cudaMemcpyDeviceToHost);
}
```

holder<T> implementations

```
template <typename T>
void
holder<T>::put(const T* hostmem, std::size_t s)
{
  if (s != size_)
    resize(s);
  cudaMemcpy(devmem, hostmem, s * sizeof(T),
             cudaMemcpyHostToDevice);
}
```

```
template <typename T>
void
holder<T>::get(T* hostmem)
{
  cudaMemcpy(hostmem, devmem, size_ * sizeof(T),
             cudaMemcpyDeviceToHost);
}
```

numpy arrays

```
>>> a
array([[  1.,   2.,   3.,   4.,   5.],
       [  6.,   7.,   8.,   9.,  10.],
       [ 11.,  12.,  13.,  14.,  15.],
       [ 16.,  17.,  18.,  19.,  20.]])
```

numpy arrays

```
>>> a
array([[  1.,   2.,   3.,   4.,   5.],
       [  6.,   7.,   8.,   9.,  10.],
       [ 11.,  12.,  13.,  14.,  15.],
       [ 16.,  17.,  18.,  19.,  20.]])


>>> a[1,:]
array([  6.,   7.,   8.,   9.,  10.])

>>> a[3,::-1]
array([ 20.,  19.,  18.,  17.,  16.])

>>> a[:,1]
array([  2.,   7.,  12.,  17.])
```

numpy arrays

```
>>> a
array([[  1.,   2.,   3.,   4.,   5.],
       [  6.,   7.,   8.,   9.,  10.],
       [ 11.,  12.,  13.,  14.,  15.],
       [ 16.,  17.,  18.,  19.,  20.]])


>>> a[1,:]
array([  6.,   7.,   8.,   9.,  10.])

>>> a[3,::-1]
array([ 20.,  19.,  18.,  17.,  16.])

>>> a[:,1]
array([  2.,   7.,  12.,  17.])
```

numpy arrays

```
>>> a
array([[  1.,   2.,   3.,   4.,   5.],
       [  6.,   7.,   8.,   9.,  10.],
       [ 11.,  12.,  13.,  14.,  15.],
       [ 16.,  17.,  18.,  19.,  20.]])


>>> a[1,:]
array([  6.,   7.,   8.,   9.,  10.])

>>> a[3,::-1]
array([ 20.,  19.,  18.,  17.,  16.])

>>> a[:,1]
array([  2.,   7.,  12.,  17.])
```

numpy arrays

```
>>> a
array([[  1.,   2.,   3.,   4.,   5.],
       [  6.,   7.,   8.,   9.,  10.],
       [ 11.,  12.,  13.,  14.,  15.],
       [ 16.,  17.,  18.,  19.,  20.]])


>>> a[1:3, 1:4]
array([[  7.,   8.,   9.],
       [ 12.,  13.,  14.]])

>>> a[2:0:-1, 3:0:-1]
array([[  9.,   8.,   7.],
       [ 14.,  13.,  12.]])
```

numpy arrays

```
>>> a
array([[  1.,   2.,   3.,   4.,   5.],
       [  6.,   7.,   8.,   9.,  10.],
       [ 11.,  12.,  13.,  14.,  15.],
       [ 16.,  17.,  18.,  19.,  20.]])


>>> a[1:3, 1:4]
array([[  7.,   8.,   9.],
       [ 12.,  13.,  14.]])

>>> a[2:0:-1, 3:0:-1]
array([[  9.,   8.,   7.],
       [ 14.,  13.,  12.]])
```

numpy arrays

```
>>> a
array([[  1.,   2.,   3.,   4.,   5.],
       [  6.,   7.,   8.,   9.,  10.],
       [ 11.,  12.,  13.,  14.,  15.],
       [ 16.,  17.,  18.,  19.,  20.]])


>>> a[1:3, 1:4]
array([[  7.,   8.,   9.],
       [ 12.,  13.,  14.]])

>>> a[2:0:-1, 3:0:-1]
array([[  9.,   8.,   7.],
       [ 14.,  13.,  12.]])


>>> b = a      # shares data
>>> b = a[:]   # copies data
```

Kamasu arrays

```
using resophonic::kamasu::_;

array<float> a(m,n)

array<float> b = a(index_range(_,_), index_range(2));

array<float> c = a(index_range(_,_,-1), index_range(_,_,-1));

float f = c(0,0); // a(9,9)

c = a;          // shares data
c = a.copy()    // copies data
```

Kamasu arrays

```
using resophonic::kamasu::_;

array<float> a(m,n)

array<float> b = a(index_range(_,_), index_range(2));

array<float> c = a(index_range(_,_,-1), index_range(_,_,-1));

float f = c(0,0); // a(9,9)

c = a;         // shares data
c = a.copy()   // copies data
```

Kamasu arrays

```
using resophonic::kamasu::_;

array<float> a(m,n)

array<float> b = a(index_range(_,_), index_range(2));

array<float> c = a(index_range(_,_,-1), index_range(_,_,-1));

float f = c(0,0); // a(9,9)

c = a;         // shares data
c = a.copy()   // copies data
```

Kamasu arrays

```
using resophonic::kamasu::_;

array<float> a(m,n)

array<float> b = a(index_range(_,_), index_range(2));

array<float> c = a(index_range(_,_,-1), index_range(_,_,-1));

float f = c(0,0); // a(9,9)

c = a;          // shares data
c = a.copy()    // copies data
```

Kamasu array metadata

```
struct view_params {
  std::size_t dims[KAMASU_MAX_ARRAY_DIM];
  std::size_t factors[KAMASU_MAX_ARRAY_DIM];
  int strides[KAMASU_MAX_ARRAY_DIM];

  offset_t offset;
  std::size_t linear_size;
  unsigned nd;
};

template <typename T>
struct array_impl
{
  view_params vp;
  shared_ptr<holder<T> > data;
};
```

Kamasu array metadata

```
struct view_params {
  std::size_t dims[KAMASU_MAX_ARRAY_DIM];
  std::size_t factors[KAMASU_MAX_ARRAY_DIM];
  int strides[KAMASU_MAX_ARRAY_DIM];

  offset_t offset;
  std::size_t linear_size;
  unsigned nd;
};

template <typename T>
struct array_impl
{
  view_params vp;
  shared_ptr<holder<T> > data;
};
```

To/from device transfer

```
namespace rk = resophonic::kamasu;

rk::array<float>                  a, b;
std::vector<float>                vec1(10), vec2(10);
ublas::matrix<float, column_major> mat1(31,14), mat2(14,31);

a << vec1;
b << vec2;
a = rk::dot(a, b);
a >> vec1;

a << mat1;
b << mat2;

a = a * b;

a >> mat1;
```

To/from device transfer

```
namespace rk = resophonic::kamasu;

rk::array<float>                   a, b;
std::vector<float>                 vec1(10), vec2(10);
ublas::matrix<float, column_major> mat1(31,14), mat2(14,31);

a << vec1;
b << vec2;
a = rk::dot(a, b);
a >> vec1;

a << mat1;
b << mat2;

a = a * b;

a >> mat1;
```

To/from device transfer

```
namespace rk = resophonic::kamasu;

rk::array<float>                   a, b;
std::vector<float>                 vec1(10), vec2(10);
ublas::matrix<float, column_major> mat1(31,14), mat2(14,31);

a << vec1;
b << vec2;
a = rk::dot(a, b);
a >> vec1;

a << mat1;
b << mat2;

a = a * b;

a >> mat1;
```

To/from device transfer

```
namespace rk = resophonic::kamasu;

rk::array<float>                   a, b;
std::vector<float>                 vec1(10), vec2(10);
ublas::matrix<float, column_major> mat1(31,14), mat2(14,31);

a << vec1;
b << vec2;
a = rk::dot(a, b);
a >> vec1;

a << mat1;
b << mat2;

a = a * b;

a >> mat1;
```

To/from device transfer

```
namespace rk = resophonic::kamasu;

rk::array<float>                   a, b;
std::vector<float>                 vec1(10), vec2(10);
ublas::matrix<float, column_major> mat1(31,14), mat2(14,31);

a << vec1;
b << vec2;
a = rk::dot(a, b);
a >> vec1;

a << mat1;
b << mat2;

a = a * b;

a >> mat1;
```

To/from device transfer

```
namespace rk = resophonic::kamasu;

rk::array<float>                   a, b;
std::vector<float>                 vec1(10), vec2(10);
ublas::matrix<float, column_major> mat1(31,14), mat2(14,31);

a << vec1;
b << vec2;
a = rk::dot(a, b);
a >> vec1;

a << mat1;
b << mat2;

a = a * b;

a >> mat1;
```

To/from device transfer

```
namespace rk = resophonic::kamasu;

rk::array<float>                     a, b;
std::vector<float>                   vec1(10), vec2(10);
ublas::matrix<float, column_major> mat1(31,14), mat2(14,31);

a << vec1;
b << vec2;
a = rk::dot(a, b);
a >> vec1;

a << mat1;
b << mat2;

a = a * b;

a >> mat1;
```

To/from device transfer

```
namespace rk = resophonic::kamasu;

rk::array<float>                   a, b;
std::vector<float>                 vec1(10), vec2(10);
ublas::matrix<float, column_major> mat1(31,14), mat2(14,31);

a << vec1;
b << vec2;
a = rk::dot(a, b);
a >> vec1;

a << mat1;
b << mat2;

a = a * b;  // calls CUBLAS sgemm()

a >> mat1;
```

To/from device transfer

```
namespace rk = resophonic::kamasu;

rk::array<float>                   a, b;
std::vector<float>                 vec1(10), vec2(10);
ublas::matrix<float, column_major> mat1(31,14), mat2(14,31);

a << vec1;
b << vec2;
a = rk::dot(a, b);
a >> vec1;

a << mat1;
b << mat2;

a = a * b;

a >> mat1;
```

Conventional greedy “old politics”

```
rk::array<float> operator+(const rk::array<float>& a, float f)

rk::array<float> a(10), b(10);
a = b + 7.0f;
```

Conventional greedy “old politics”

```
rk::array<float> operator+(const rk::array<float>& a, float f)

rk::array<float> a(10), b(10);
a = b + 7.0f;
```

b + 7.0f;

```
expr<
  tag::plus,
  list2<
    expr<
      tag::terminal,
      array<float> const&,
    >,
    expr<
      tag::terminal,
      float const&,
    >
  >,
>
```

b + 7.0f;

```
expr<
  tag::plus,
  list2<
    expr<
      tag::terminal,
      array<float> const&,
    >,
    expr<
      tag::terminal,
      float const&,
    >
  >,
>
```

b + 7.0f;

```
expr<
  tag::plus,
  list2<
    expr<
      tag::terminal,
      array<float> const&,
    >,
    expr<
      tag::terminal,
      float const&,
    >
  >,
>
```

b + 7.0f;

```
expr<
  tag::plus,
  list2<
    expr<
      tag::terminal,
      array<float> const&,
    >,
    expr<
      tag::terminal,
      float const&,
    >
  >,
>
```

b + 7.0f;

```
expr<
  tag::plus,
  list2<
    expr<
      tag::terminal,
      array<float> const&,
    >,
    expr<
      tag::terminal,
      float const&,
    >
  >,
>
```

b + 7.0f;

```
expr<
  tag::plus,
  list2<
    expr<
      tag::terminal,
      array<float> const&,
    >,
    expr<
      tag::terminal,
      float const&,
    >
  >,
>
```

b + 7.0f;

```
expr<
  tag::plus,
  list2<
    expr<
      tag::terminal,
      array<float> const&,
    >,
    expr<
      tag::terminal,
      float const&,
    >
  >,
>
```

b + **7.0f**;

```
expr<
  tag::plus,
  list2<
    expr<
      tag::terminal,
      array<float> const&,
    >,
    expr<
      tag::terminal,
      float const&,
    >
  >,
>
```

b + **7.0f**;

```
expr<
  tag::plus,
  list2<
    expr<
      tag::terminal,
      array<float> const&,
    >,
    expr<
      tag::terminal,
      float const&,
    >
  >,
>
```

b + **7.0f**;

```
expr<
  tag::plus,
  list2<
    expr<
      tag::terminal,
      array<float> const&,
    >,
    expr<
      tag::terminal,
      float const&,
    >
  >,
>
```

b + **7.0f**;

```
expr<
  tag::plus,
  list2<
    expr<
      tag::terminal,
      array<float> const&,
    >,
    expr<
      tag::terminal,
      float const&,
    >
  >,
>
```

b + 7.0f;

```
boost::proto::exprns_::expr<
   boost::proto::tag::plus,
   boost::proto::argsns_::list2<
     boost::proto::exprns_::expr<
       boost::proto::tag::terminal,
       boost::proto::argsns_::term<array<float> >,
       0
     >,
     boost::proto::exprns_::expr<
       boost::proto::tag::terminal,
       boost::proto::argsns_::term<int const&>,
       0
     >
   >,
   2
>
```

b + 7.0f;

```
exprns_::expr<
   tag::plus,
   argsns_::list2<
     exprns_::expr<
       tag::terminal,
       argsns_::term<array<float> >,
       0
     >,
     exprns_::expr<
       tag::terminal,
       argsns_::term<int const&>,
       0
     >
   >,
   2
>
```

b + 7.0f;

```
expr<
   tag::plus,
   list2<
     expr<
       tag::terminal,
       term<array<float> >,
       0
     >,
     expr<
       tag::terminal,
       term<int const&>,
       0
     >
   >,
   2
>
```

b + 7.0f;

```
expr<
   tag::plus,
   list2<
     expr<
       tag::terminal,
       term<array<float> >,
       0
     >,
     expr<
       tag::terminal,
       term<int const&>,
       0
     >
   >,
   2
>
```

b + 7.0f;

```
expr<
   tag::plus,
   list2<
     expr<
       tag::terminal,
       term<array<float> >,
       0   // arity
     >,
     expr<
       tag::terminal,
       term<int const&>,
       0
     >
   >,
   2
>
```

b + 7.0f;

```
expr<
   tag::plus,
   list2<
     expr<
       tag::terminal,
       term<array<float> >,
       0
     >,
     expr<
       tag::terminal,
       term<int const&>,
       0
     >
   >,
   2
>
```

an array is an expression

```
proto::exprns_::expr<
  proto::tag::terminal,
  proto::argsns_::term<T>,
  0
>

proto::terminal<T>::type

namespace bp = boost::proto;

template <typename T>
struct array_impl { /* data goes here */ };

template <typename T>
struct array : proto::terminal<array_impl<T> >::type
{ ... };
```

an array is an expression

```
proto::exprns_::expr<
  proto::tag::terminal,
  proto::argsns_::term<T>,
  0
>

proto::terminal<T>::type

namespace bp = boost::proto;

template <typename T>
struct array_impl { /* data goes here */ };

template <typename T>
struct array : proto::terminal<array_impl<T> >::type
{ ... };
```

an array is an expression

```
proto::exprns_::expr<
  proto::tag::terminal,
  proto::argsns_::term<T>,
  0
>

proto::terminal<T>::type

namespace bp = boost::proto;

template <typename T>
struct array_impl { /* data goes here */ };

template <typename T>
struct array : proto::terminal<array_impl<T> >::type
{ ... };
```

an array is an expression

```
proto::exprns_::expr<
  proto::tag::terminal,
  proto::argsns_::term<T>,
  0
>

proto::terminal<T>::type

namespace bp = boost::proto;

template <typename T>
struct array_impl { /* data goes here */ };

template <typename T>
struct array : proto::terminal<array_impl<T> >::type
{ ... };
```

an array is an expression

```
proto::exprns_::expr<
  proto::tag::terminal,
  proto::argsns_::term<T>,
  0
>

proto::terminal<T>::type

namespace bp = boost::proto;

template <typename T>
struct array_impl { /* data goes here */ };

template <typename T>
struct array : proto::terminal<array_impl<T> >::type
{ ... };
```

an array is an expression

```
proto::exprns_::expr<
  proto::tag::terminal,
  proto::argsns_::term<T>,
  0
>
```

```
proto::terminal<T>::type
```

```
namespace bp = boost::proto;

template <typename T>
struct array_impl { /* data goes here */ };

template <typename T>
struct array : proto::terminal<array_impl<T> >::type
{ ... };
```

an array is an expression

```
proto::exprns_::expr<
  proto::tag::terminal,
  proto::argsns_::term<T>,
  0
>

proto::terminal<T>::type

namespace bp = boost::proto;

template <typename T>
struct array_impl { /* data goes here */ };

template <typename T>
struct array : proto::terminal<array_impl<T> >::type
{ ... };
```

an array is an expression

```
proto::exprns_::expr<
  proto::tag::terminal,
  proto::argsns_::term<T>,
  0
>

proto::terminal<T>::type

namespace bp = boost::proto;

template <typename T>
struct array_impl { /* data goes here */ };

template <typename T>
struct array : proto::terminal<array_impl<T> >::type
{ ... };
```

Hello World

```
struct array_impl { };

struct array : bp::terminal<array_impl>::type
{
  template <typename Expr>
  void operator=(const Expr& expr)
  {
    std::cout « name_of(expr);
  }
};

array a, b;
a = b + 7.0f;
```

```
boost::proto::exprns_::expr<boost::proto::tag::plus,
boost::proto::argsns_::list2<array&, boost::proto::e
xprns_::expr<boost::proto::tag::terminal, boost::pro
to::argsns_::term<float const&>, 0l> >, 2l>
```

Hello World

```
struct array_impl { };

struct array : bp::terminal<array_impl>::type
{
  template <typename Expr>
  void operator=(const Expr& expr)
  {
    std::cout « name_of(expr);
  }
};

array a, b;
a = b + 7.0f;
```

```
boost::proto::exprns_::expr<boost::proto::tag::plus,
boost::proto::argsns_::list2<array&, boost::proto::e
xprns_::expr<boost::proto::tag::terminal, boost::pro
to::argsns_::term<float const&>, 0l> >, 2l>
```

Hello World

```
struct array_impl { };

struct array : bp::terminal<array_impl>::type
{
  template <typename Expr>
  void operator=(const Expr& expr)
  {
    std::cout « name_of(expr);
  }
};

array a, b;
a = b + 7.0f;
```

```
boost::proto::exprns_::expr<boost::proto::tag::plus,
boost::proto::argsns_::list2<array&, boost::proto::e
xprns_::expr<boost::proto::tag::terminal, boost::pro
to::argsns_::term<float const&>, 0l> >, 2l>
```

Hello World

```
struct array_impl { };

struct array : bp::terminal<array_impl>::type
{
  template <typename Expr>
  void operator=(const Expr& expr)
  {
    std::cout « name_of(expr);
  }
};

array a, b;
a = b + 7.0f;
```

```
boost::proto::exprns_::expr<boost::proto::tag::plus,
boost::proto::argsns_::list2<array&, boost::proto::e
xprns_::expr<boost::proto::tag::terminal, boost::pro
to::argsns_::term<float const&>, 0l> >, 2l>
```

display_expr

```
std::ostream& operator«(std::ostream& s, array_impl)
{
  return s « "array_impl";
}

struct array : bp::terminal<array_impl>::type
{
  template <typename Expr>
  void operator=(const Expr& expr)
  {
    std::cout « bp::display_expr(expr);
  }
};

array a, b;
a = b + 7.0f;
```

display_expr

```
std::ostream& operator«(std::ostream& s, array_impl)
{
  return s « "array_impl";
}

struct array : bp::terminal<array_impl>::type
{
  template <typename Expr>
  void operator=(const Expr& expr)
  {
    std::cout « bp::display_expr(expr);
  }
};

array a, b;
a = b + 7.0f;
```

display_expr

```
std::ostream& operator«(std::ostream& s, array_impl)
{
  return s « "array_impl";
}

struct array : bp::terminal<array_impl>::type
{
  template <typename Expr>
  void operator=(const Expr& expr)
  {
    std::cout « bp::display_expr(expr);
  }
};

array a, b;
a = b + 7.0f;
```

display_expr

```
std::ostream& operator«(std::ostream& s, array_impl)
{
  return s « "array_impl";
}

struct array : bp::terminal<array_impl>::type
{
  template <typename Expr>
  void operator=(const Expr& expr)
  {
    std::cout « bp::display_expr(expr);
  }
};

array a, b;
a = b + 7.0f;
```

display_expr

```
std::ostream& operator«(std::ostream& s, array_impl)
{
  return s « "array_impl";
}

struct array : bp::terminal<array_impl>::type
{
  template <typename Expr>
  void operator=(const Expr& expr)
  {
    std::cout « bp::display_expr(expr);
  }
};

array a, b;
a = b + 7.0f;
```

display_expr

```
std::ostream& operator«(std::ostream& s, array_impl)
{
  return s « "array_impl";
}

struct array : bp::terminal<array_impl>::type
{
  template <typename Expr>
  void operator=(const Expr& expr)
  {
    std::cout « bp::display_expr(expr);
  }
};
```

```
array a, b;
a = b + 7.0f;
```

```
plus(
    terminal(array_impl)
  , terminal(7)
)
```

Operator tags

```
namespace boost {
  namespace proto {
    namespace tag {
      struct terminal;
      struct unary_plus;
      struct negate;
      struct dereference;
      struct complement;
      struct address_of;
      struct logical_not;
      struct pre_inc;
      struct pre_dec;
      struct post_inc;
      struct post_dec;
      struct shift_left;
      struct shift_right;
      struct multiplies;
      struct divides;
      struct modulus;
```

Operator tags

```
namespace boost {
  namespace proto {
    namespace tag {
      struct plus;
      struct minus;
      struct less;
      struct greater;
      struct less_equal;
      struct greater_equal;
      struct equal_to;
      struct not_equal_to;
      struct logical_or;
      struct logical_and;
      struct bitwise_and;
      struct bitwise_or;
      struct bitwise_xor;
      struct comma;
      struct mem_ptr;
      struct assign;
      struct shift_left_assign;
```

Operator tags

```
namespace boost {
  namespace proto {
    namespace tag {
      struct shift_right_assign;
      struct multiplies_assign;
      struct divides_assign;
      struct modulus_assign;
      struct plus_assign;
      struct minus_assign;
      struct bitwise_and_assign;
      struct bitwise_or_assign;
      struct bitwise_xor_assign;
      struct subscript;
      struct if_else_;
      struct function;
```

Operator Swine Flu

```
trespasser s, t;   array m, n;
m = s & ~n >>= std::cout % m->*t;

shift_right_assign(
    bitwise_and(
        terminal(trespasser)
      , complement(
            terminal(array_impl)
        )
    )
  , modulus(
        terminal(0x607068)
      , mem_ptr(
            terminal(array_impl)
          , terminal(trespasser)
        )
    )
)
```

Operator Swine Flu

```
trespasser s, t;   array m, n;
m = s & ~n >>= std::cout % m->*t;

shift_right_assign(
    bitwise_and(
        terminal(trespasser)
      , complement(
            terminal(array_impl)
        )
    )
  , modulus(
        terminal(0x607068)
      , mem_ptr(
            terminal(array_impl)
          , terminal(trespasser)
        )
    )
)
```

Operator Swine Flu

```
trespasser s, t;   array m, n;
m = s & ~n >>= std::cout % m->*t;

shift_right_assign(
    bitwise_and(
        terminal(trespasser)
      , complement(
            terminal(array_impl)
        )
    )
  , modulus(
        terminal(0x607068)
      , mem_ptr(
            terminal(array_impl)
          , terminal(trespasser)
        )
    )
)
```

Operator Swine Flu

```
trespasser s, t;   array m, n;
m = s & ~n >>= std::cout % m->*t;

shift_right_assign(
    bitwise_and(
        terminal(trespasser)
      , complement(
            terminal(array_impl)
        )
    )
  , modulus(
        terminal(0x607068)
      , mem_ptr(
            terminal(array_impl)
          , terminal(trespasser)
        )
    )
)
```

Operator Swine Flu

```
trespasser s, t;   array m, n;
m = s & ~n >>= std::cout % m->*t;

shift_right_assign(
    bitwise_and(
        terminal(trespasser)
      , complement(
            terminal(array_impl)
        )
    )
  , modulus(
        terminal(0x607068)
      , mem_ptr(
            terminal(array_impl)
          , terminal(trespasser)
        )
    )
)
```

Operator Swine Flu

```
trespasser s, t;   array m, n;
m = s & ~n >>= std::cout % m->*t;

shift_right_assign(
    bitwise_and(
        terminal(trespasser)
      , complement(
            terminal(array_impl)
        )
    )
  , modulus(
        terminal(0x607068)
      , mem_ptr(
            terminal(array_impl)
          , terminal(trespasser)
        )
    )
)
```

Operator Swine Flu

```
trespasser s, t;   array m, n;
m = s & ~n >>= std::cout % m->*t;

shift_right_assign(
    bitwise_and(
        terminal(trespasser)
      , complement(
            terminal(array_impl)
        )
    )
  , modulus(
        terminal(0x607068)
      , mem_ptr(
            terminal(array_impl)
          , terminal(trespasser)
        )
    )
)
```

Operator Swine Flu

```
trespasser s, t;   array m, n;
m = s & ~n >>= std::cout % m->*t;

shift_right_assign(
    bitwise_and(
        terminal(trespasser)
      , complement(
            terminal(array_impl)
        )
    )
  , modulus(
        terminal(0x607068)
      , mem_ptr(
            terminal(array_impl)
          , terminal(trespasser)
        )
    )
)
```

Operator Swine Flu

```
trespasser s, t;   array m, n;
m = s & ~n >>= std::cout % m->*t;

shift_right_assign(
    bitwise_and(
        terminal(trespasser)
      , complement(
            terminal(array_impl)
        )
    )
  , modulus(
        terminal(0x607068)
      , mem_ptr(
            terminal(array_impl)
          , terminal(trespasser)
        )
    )
)
```

Operator Swine Flu

```
trespasser s, t;   array m, n;
m = s & ~n >>= std::cout % m->*t;

shift_right_assign(
    bitwise_and(
        terminal(trespasser)
      , complement(
            terminal(array_impl)
        )
    )
  , modulus(
        terminal(0x607068)
      , mem_ptr(
            terminal(array_impl)
          , terminal(trespasser)
        )
    )
)
```

Operator Swine Flu

```
trespasser s, t;   array m, n;
m = s & ~n >>= std::cout % m->*t;

shift_right_assign(
    bitwise_and(
        terminal(trespasser)
      , complement(
            terminal(array_impl)
        )
    )
  , modulus(
        terminal(0x607068)
      , mem_ptr(
            terminal(array_impl)
          , terminal(trespasser)
        )
    )
)
```

Operator Swine Flu

```
trespasser s, t;   array m, n;
m = s & ~n >>= std::cout % m->*t;

shift_right_assign(
    bitwise_and(
        terminal(trespasser)
      , complement(
            terminal(array_impl)
        )
    )
  , modulus(
        terminal(0x607068)
      , mem_ptr(
            terminal(array_impl)
          , terminal(trespasser)
        )
    )
)
```

Transforms

```
struct array_impl { /* pointers, strides, etc. */ };
struct array : bp::terminal<array_impl>::type
{ /* ... */ };

array a;

array_impl& aimpl =
   a.child0;           // via base class' member
   bp::value(a);       // free function
   bp::_value()(a);    // instance of _value
   bp::_child0()(a);   // instance of _child0
   bp::child_c<0>(a);
   bp::child<mpl::long_<0> >(a);
   bp::_left()(a);     // instance of _left
   bp::left(a);        // left function
```

Transforms

```
struct array_impl { /* pointers, strides, etc. */ };
struct array : bp::terminal<array_impl>::type
{ /* ... */ };

array a;

array_impl& aimpl =
   a.child0;            // via base class’ member
   bp::value(a);        // free function
   bp::_value()(a);     // instance of _value
   bp::_child0()(a);    // instance of _child0
   bp::child_c<0>(a);
   bp::child<mpl::long_<0> >(a);
   bp::_left()(a);      // instance of _left
   bp::left(a);         // left function
```

Transforms

```
struct array_impl { /* pointers, strides, etc. */ };
struct array : bp::terminal<array_impl>::type
{ /* ... */ };

array a;

array_impl& aimpl =
   a.child0;            // via base class' member
   bp::value(a);        // free function
   bp::_value()(a);     // instance of _value
   bp::_child0()(a);    // instance of _child0
   bp::child_c<0>(a);
   bp::child<mpl::long_<0> >(a);
   bp::_left()(a);      // instance of _left
   bp::left(a);         // left function
```

Transforms

```
struct array_impl { /* pointers, strides, etc. */ };
struct array : bp::terminal<array_impl>::type
{ /* ... */ };

array a;

array_impl& aimpl =
   a.child0;          // via base class' member
   bp::value(a);      // free function
   bp::_value()(a);   // instance of _value
   bp::_child0()(a);  // instance of _child0
   bp::child_c<0>(a);
   bp::child<mpl::long_<0> >(a);
   bp::_left()(a);    // instance of _left
   bp::left(a);       // left function
```

Transforms

```
struct array_impl { /* pointers, strides, etc. */ };
struct array : bp::terminal<array_impl>::type
{ /* ... */ };

array a;

array_impl& aimpl =
   a.child0;             // via base class' member
   bp::value(a);         // free function
   bp::_value()(a);      // instance of _value
   bp::_child0()(a);     // instance of _child0
   bp::child_c<0>(a);
   bp::child<mpl::long_<0> >(a);
   bp::_left()(a);       // instance of _left
   bp::left(a);          // left function
```

Transforms

```
struct array_impl { /* pointers, strides, etc. */ };
struct array : bp::terminal<array_impl>::type
{ /* ... */ };

array a;

array_impl& aimpl =
   a.child0;            // via base class’ member
   bp::value(a);        // free function
   bp::_value()(a);     // instance of _value
   bp::_child0()(a);    // instance of _child0
   bp::child_c<0>(a);
   bp::child<mpl::long_<0> >(a);
   bp::_left()(a);      // instance of _left
   bp::left(a);         // left function
```

Transforms

```
struct array_impl { /* pointers, strides, etc. */ };
struct array : bp::terminal<array_impl>::type
{ /* ... */ };

array a;

array_impl& aimpl =
   a.child0;          // via base class' member
   bp::value(a);      // free function
   bp::_value()(a);   // instance of _value
   bp::_child0()(a);  // instance of _child0
   bp::child_c<0>(a);
   bp::child<mpl::long_<0> >(a);
   bp::_left()(a);    // instance of _left
   bp::left(a);       // left function
```

Transforms

```
struct array_impl { /* pointers, strides, etc. */ };
struct array : bp::terminal<array_impl>::type
{ /* ... */ };

array a;

array_impl& aimpl =
   a.child0;             // via base class’ member
   bp::value(a);         // free function
   bp::_value()(a);      // instance of _value
   bp::_child0()(a);     // instance of _child0
   bp::child_c<0>(a);
   bp::child<mpl::long_<0> >(a);
   bp::_left()(a);       // instance of _left
   bp::left(a);          // left function
```

Transforms

```
struct array_impl { /* pointers, strides, etc. */ };
struct array : bp::terminal<array_impl>::type
{ /* ... */ };

array a;

array_impl& aimpl =
   a.child0;          // via base class’ member
   bp::value(a);      // free function
   bp::_value()(a);   // instance of _value
   bp::_child0()(a);  // instance of _child0
   bp::child_c<0>(a);
   bp::child<mpl::long_<0> >(a);
   bp::_left()(a);    // instance of _left
   bp::left(a);       // left function
```

Transforms

```
struct array_impl { /* pointers, strides, etc. */ };
struct array : bp::terminal<array_impl>::type
{ /* ... */ };

array a;

array_impl& aimpl =
   a.child0;          // via base class' member
   bp::value(a);      // free function
   bp::_value()(a);   // instance of _value
   bp::_child0()(a);  // instance of _child0
   bp::child_c<0>(a);
   bp::child<mpl::long_<0> >(a);
   bp::_left()(a);    // instance of _left
   bp::left(a);       // left function
```

Transforms

```
struct array_impl { /* pointers, strides, etc. */ };
struct array : bp::terminal<array_impl>::type
{ /* ... */ };

array a;

array_impl& aimpl =
   a.child0;          // via base class’ member
   bp::value(a);      // free function
   bp::_value()(a);   // instance of _value
   bp::_child0()(a);  // instance of _child0
   bp::child_c<0>(a);
   bp::child<mpl::long_<0> >(a);
   bp::_left()(a);    // instance of _left
   bp::left(a);       // left function
```

Transforms

```
struct array_impl { /* pointers, strides, etc. */ };
struct array : bp::terminal<array_impl>::type
{ /* ... */ };

array a;

array_impl& aimpl =
   a.child0;          // via base class’ member
   bp::value(a);      // free function
   bp::_value()(a);   // instance of _value
   bp::_child0()(a);  // instance of _child0
   bp::child_c<0>(a);
   bp::child<mpl::long_<0> >(a);
   bp::_left()(a);    // instance of _left
   bp::left(a);       // left function
```

Transforms

```
struct array_impl { /* pointers, strides, etc. */ };
struct array : bp::terminal<array_impl>::type
{ /* ... */ };

array a;

array_impl& aimpl =
   a.child0;          // via base class' member
   bp::value(a);      // free function
   bp::_value()(a);   // instance of _value
   bp::_child0()(a);  // instance of _child0
   bp::child_c<0>(a);
   bp::child<mpl::long_<0> >(a);
   bp::_left()(a);    // instance of _left
   bp::left(a);       // left function
```

More transforms

```
struct array_impl { float value; };
struct array : bp::terminal<array_impl>::type { ... };
array t;

t + 2
```

```
expr<
  tag::plus
, list2<
    array&
  , expr<
      tag::terminal
    , term<int const&>
    , 0
    >
  >
, 2
>
```

```
plus(
    terminal(array_impl)
  , terminal(2)
)
```

More transforms

```
struct array_impl { float value; };
struct array : bp::terminal<array_impl>::type { ... };
array t;
```

```
t + 2
```

```
expr<
  tag::plus
, list2<
    array&
  , expr<
      tag::terminal
    , term<int const&>
    , 0
    >
  >
, 2
>
```

```
plus(
    terminal(array_impl)
  , terminal(2)
)
```

More transforms

```
struct array_impl { float value; };
struct array : bp::terminal<array_impl>::type { ... };
array t;

t + 2
```

```
expr<
  tag::plus
, list2<
    array&
  , expr<
      tag::terminal
    , term<int const&>
    , 0
    >
  >
, 2
>
```

```
plus(
    terminal(array_impl)
  , terminal(2)
)
```

More transforms

```
struct array_impl { float value; };
struct array : bp::terminal<array_impl>::type { ... };
array t;
```

```
bp::display_expr(t + 2)
```

```
expr<
  tag::plus
, list2<
    array&
  , expr<
      tag::terminal
    , term<int const&>
    , 0
    >
  >
, 2
>
```

```
plus(
    terminal(array_impl)
  , terminal(2)
)
```

More transforms

```
struct array_impl { float value; };
struct array : bp::terminal<array_impl>::type { ... };
array t;

bp::display_expr(t + 2)
```

```
expr<
  tag::plus
, list2<
    array&
  , expr<
      tag::terminal
    , term<int const&>
    , 0
    >
  >
, 2
>
```

```
plus(
    terminal(array_impl)
  , terminal(2)
)
```

More transforms

```
struct array_impl { float value; };
struct array : bp::terminal<array_impl>::type { ... };
array t;

bp::display_expr(t + 2)
```

```
plus(
    terminal(array_impl)
  , terminal(2)
)
```

More transforms

```
struct array_impl { float value; };
struct array : bp::terminal<array_impl>::type { ... };
array t;

bp::display_expr(bp::child1(t + 2))
```

```
terminal(2)
```

More transforms

```
struct array_impl { float value; };
struct array : bp::terminal<array_impl>::type { ... };
array t;

std::cout « bp::value(bp::child1(t + 2))
```

```
2
```

More transforms

```
struct array_impl { float value; };
struct array : bp::terminal<array_impl>::type { ... };
array t;

bp::value(bp::right(bp::left(a++->* 7 /= 4)));

divides_assign(
    mem_ptr(
        post_inc(
            terminal(array_impl)
        )
      , terminal(7)
    )
  , terminal(4)
)
```

More transforms

```
struct array_impl { float value; };
struct array : bp::terminal<array_impl>::type { ... };
array t;

bp::value(bp::right(bp::left(a++->* 7 /= 4)));

divides_assign(
    mem_ptr(
        post_inc(
            terminal(array_impl)
        )
      , terminal(7)
    )
  , terminal(4)
)
```

More transforms

```
struct array_impl { float value; };
struct array : bp::terminal<array_impl>::type { ... };
array t;

bp::value(bp::right(bp::left(a++->* 7 /= 4)));
```

```
divides_assign(
    mem_ptr(
        post_inc(
            terminal(array_impl)
        )
      , terminal(7)
    )
  , terminal(4)
)
```

More transforms

```
struct array_impl { float value; };
struct array : bp::terminal<array_impl>::type { ... };
array t;

bp::value(bp::right(bp::left(a++->* 7 /= 4)));

divides_assign(
    mem_ptr(
        post_inc(
            terminal(array_impl)
        )
      , terminal(7)
    )
  , terminal(4)
)
```

More transforms

```
struct array_impl { float value; };
struct array : bp::terminal<array_impl>::type { ... };
array t;

bp::value(bp::right(bp::left(a++->* 7 /= 4)));

divides_assign(
    mem_ptr(
        post_inc(
            terminal(array_impl)
        )
      , terminal(7)
    )
  , terminal(4)
)
```

More transforms

```
struct array_impl { float value; };
struct array : bp::terminal<array_impl>::type { ... };
array t;

bp::value(bp::right(bp::left(a++->* 7 /= 4)));

divides_assign(
    mem_ptr(
        post_inc(
            terminal(array_impl)
        )
      , terminal(7)
    )
  , terminal(4)
)
```

More transforms

```
struct array_impl { float value; };
struct array : bp::terminal<array_impl>::type { ... };
array t;

bp::value(bp::right(bp::left(a++->* 7 /= 4)));

divides_assign(
    mem_ptr(
        post_inc(
            terminal(array_impl)
        )
      , terminal(7)
    )
  , terminal(4)
)
```

More transforms

```
struct array_impl { float value; };
struct array : bp::terminal<array_impl>::type { ... };
array t;
```

```
bp::value(bp::right(bp::left(a++->* 7 /= 4)));
```

```
divides_assign(
    mem_ptr(
        post_inc(
            terminal(array_impl)
        )
      , terminal(7)
    )
  , terminal(4)
)
```

More transforms

```
struct array_impl { float value; };
struct array : bp::terminal<array_impl>::type { ... };
array t;

bp::value(bp::right(bp::left(a++->* 7 /= 4)));

divides_assign(
    mem_ptr(
        post_inc(
            terminal(array_impl)
        )
      , terminal(7)
    )
  , terminal(4)
)
```

Matching expressions A

```
struct Grammar
  : bp::terminal<array_impl<float> >,
{ };
```

```
template <typename Expr>
void
array::operator=(const Expr& expr)
{
  std::cout << bp::matches<Expr, Grammar>() « "\n";
}

m = n * 17.0f;   // prints ’0’
m = n;           // prints ’1’
```

Matching expressions A

```
struct Grammar
  : bp::terminal<array_impl<float> >,
{ };
```

```
template <typename Expr>
void
array::operator=(const Expr& expr)
{
  std::cout << bp::matches<Expr, Grammar>() « "\n";
}

m = n * 17.0f;    // prints ’0’
m = n;            // prints ’1’
```

Matching expressions A

```
struct Grammar
  : bp::terminal<array_impl<float> >,
{ };
```

```
template <typename Expr>
void
array::operator=(const Expr& expr)
{
  std::cout << bp::matches<Expr, Grammar>() « "\n";
}

m = n * 17.0f;   // prints ’0’
m = n;           // prints ’1’
```

Matching expressions A

```
struct Grammar
  : bp::terminal<array_impl<float> >,
{ };




template <typename Expr>
void
array::operator=(const Expr& expr)
{
  std::cout << bp::matches<Expr, Grammar>() « "\n";
}

m = n * 17.0f;   // prints ’0’
m = n;           // prints ’1’
```

Matching expressions A

```
struct Grammar
  : bp::terminal<array_impl<float> >,
{ };
```

```
template <typename Expr>
void
array::operator=(const Expr& expr)
{
  std::cout << bp::matches<Expr, Grammar>() « "\n";
}

m = n * 17.0f;    // prints '0'
m = n;            // prints '1'
```

Matching expressions

```
struct Grammar
  : bp::or_<bp::terminal<array_impl<float> >,
            bp::terminal<float>,
            bp::plus<bp::terminal<array_impl<float> >,
                     bp::terminal<float> >
            >
{ };

template <typename Expr>
void
array::operator=(const Expr& expr)
{
  std::cout << bp::matches<Expr, Grammar>() « "\n";
}

m = s & ~m >>= std::cout % n->*t;   // prints ’0’
m = n + 7.0f;                       // prints ’1’
```

Matching expressions

```
struct Grammar
  : bp::or_<bp::terminal<array_impl<float> >,
            bp::terminal<float>,
            bp::plus<bp::terminal<array_impl<float> >,
                     bp::terminal<float> >
            >
{ };

template <typename Expr>
void
array::operator=(const Expr& expr)
{
  std::cout << bp::matches<Expr, Grammar>() « "\n";
}

m = s & ~m >>= std::cout % n->*t;   // prints ’0’
m = n + 7.0f;                       // prints ’1’
```

Matching expressions

```
struct Grammar
  : bp::or_<bp::terminal<array_impl<float> >,
            bp::terminal<float>,
            bp::plus<bp::terminal<array_impl<float> >,
                     bp::terminal<float> >
            >
{ };

template <typename Expr>
void
array::operator=(const Expr& expr)
{
  std::cout << bp::matches<Expr, Grammar>() « "\n";
}

m = s & ~m >>= std::cout % n->*t;    // prints ’0’
m = n + 7.0f;                        // prints ’1’
```

Matching expressions

```
struct Grammar
  : bp::or_<bp::terminal<array_impl<float> >,
            bp::terminal<float>,
            bp::plus<bp::terminal<array_impl<float> >,
                     bp::terminal<float> >
            >
{ };

template <typename Expr>
void
array::operator=(const Expr& expr)
{
  std::cout << bp::matches<Expr, Grammar>() « "\n";
}

m = s & ~m >>= std::cout % n->*t;   // prints ’0’
m = n + 7.0f;                       // prints ’1’
```

Matching expressions

```
struct Grammar
  : bp::or_<bp::terminal<array_impl<float> >,
            bp::terminal<float>,
            bp::plus<bp::terminal<array_impl<float> >,
                     bp::terminal<float> >
            >
{ };

template <typename Expr>
void
array::operator=(const Expr& expr)
{
  std::cout << bp::matches<Expr, Grammar>() « "\n";
}

m = s & ~m >>= std::cout % n->*t;    // prints ’0’
m = n + 7.0f;                        // prints ’1’
```

Matching expressions

```
struct Grammar
  : bp::or_<bp::terminal<array_impl<float> >,
            bp::terminal<float>,
            bp::plus<bp::terminal<array_impl<float> >,
                     bp::terminal<float> >
            >
{ };

template <typename Expr>
void
array::operator=(const Expr& expr)
{
  std::cout << bp::matches<Expr, Grammar>() « "\n";
}

m = s & ~m >>= std::cout % n->*t;    // prints ’0’
m = n + 7.0f;                        // prints ’1’
```

Detecting invalid expressions

```
template <typename Expr>
void
array::operator=(const Expr& expr)
{
  // to come: do something with expr

}

m = s & ~m >>= std::cout % n->*t;
```

Detecting invalid expressions

```
template <typename Expr>
typename enable_if<bp::matches<Expr, Grammar> >::type
array::operator=(const Expr& expr)
{
  // to come: do something with expr

}

m = s & ~m >>= std::cout % n->*t;
```

Detecting invalid expressions

```
template <typename Expr>
typename enable_if<bp::matches<Expr, Grammar> >::type
array::operator=(const Expr& expr)
{
  // to come: do something with expr

}
```

```
m = s & ~m >>= std::cout % n->*t;
error: no match for ’operator=’ in ’m = boost::prot
o::exprns_::operator»= [with Left = boost::proto::
exprns_::expr<boost::proto::tag::bitwise_and, boost
::proto::argsns_::list2<boost::proto::exprns_::expr
<boost::proto::tag::terminal, boost::proto::argsns_
::term<trespasser&>, 0l>, const boost::proto::exprn
s_::expr<boost::proto::tag::complement, boost::prot
o::argsns_::list1<boost::proto::exprns_::expr<bo...
```

Detecting invalid expressions

```
template <typename Expr>
void
array::operator=(const Expr& expr)
{
  // to come: do something with expr
  BOOST_MPL_ASSERT((bp::matches<Expr, Grammar>));
}

m = s & ~m >>= std::cout % n->*t;
```

Detecting invalid expressions

```
template <typename Expr>
void
array::operator=(const Expr& expr)
{
  // to come: do something with expr
  BOOST_MPL_ASSERT((bp::matches<Expr, Grammar>));
}
```

```
m = s & ~m >>= std::cout % n->*t;
error: no matching function for call to ’assertion_
failed(mpl_::failed************ boost::proto::resul
t_of::matches<boost::proto::exprns_::expr<boost::pr
oto::tag::shift_right_assign, boost::proto::argsns_
::list2<const boost::proto::exprns_::expr<boost::pr
oto::tag::bitwise_and, boost::proto::argsns_::list2
<boost::proto::exprns_::expr<boost::proto::tag::ter
mina  ....  proto::a>, 2l>, Grammar>::************)’
```

Detecting invalid expressions

```
template <typename Expr>
void
array::operator=(const Expr& expr)
{
  // to come: do something with expr
  BOOST_MPL_ASSERT((bp::matches<Expr, Grammar>));
}
```

```
m = s & ~m >>= std::cout % n->*t;
error: no matching function for call to 'assertion_
failed(mpl_::failed************ boost::proto::resul
t_of::matches<boost::proto::exprns_::expr<boost::pr
oto::tag::shift_right_assign, boost::proto::argsns_
::list2<const boost::proto::exprns_::expr<boost::pr
oto::tag::bitwise_and, boost::proto::argsns_::list2
<boost::proto::exprns_::expr<boost::proto::tag::ter
mina  ....  proto::a>, 2l>, Grammar>::************)'
```

Quick reminder

```
int foo(double) { ... }

boost::function<int(double)> foofn = foo;
```

Quick reminder

```
int foo(double) { ... }

boost::function<int(double)> foofn = foo;
```

Quick reminder

```
int foo(double) { ... }

boost::function<int(double)> foofn = foo;


struct bar { };
bar makebar()   ...

bar();

boost::function<bar()> barfn = makebar;
```

Quick reminder

```
int foo(double) { ... }

boost::function<int(double)> foofn = foo;


struct bar { };
bar makebar()  ...

bar();

boost::function<bar()> barfn = makebar;
```

Quick reminder

```
int foo(double) { ... }

boost::function<int(double)> foofn = foo;


struct bar { };
bar makebar()  ...

bar();

boost::function<bar()> barfn = makebar;
```

Activating a grammar

```
struct ToString : bp::callable
{
  typedef std::string result_type;

  template <typename T>
  result_type operator()(const T& t)
  {
    return str(boost::format("%s @ %p") % t % &t);
  }
};

struct FloatTerminal
  : bp::when<bp::terminal<float>, ToString(bp::_value)>
{ };

bp::terminal<float>::type f = { 3.14 };
std::cout « FloatTerminal()(f);
“3.14 @ 0x7fff0d839aa0”
```

Activating a grammar

```
struct ToString : bp::callable
{
  typedef std::string result_type;

  template <typename T>
  result_type operator()(const T& t)
  {
    return str(boost::format("%s @ %p") % t % &t);
  }
};

struct FloatTerminal
  : bp::when<bp::terminal<float>, ToString(bp::_value)>
{ };

bp::terminal<float>::type f = { 3.14 };
std::cout « FloatTerminal()(f);
“3.14 @ 0x7fff0d839aa0”
```

Activating a grammar

```
struct ToString : bp::callable
{
  typedef std::string result_type;

  template <typename T>
  result_type operator()(const T& t)
  {
    return str(boost::format("%s @ %p") % t % &t);
  }
};

struct FloatTerminal
  : bp::when<bp::terminal<float>, ToString(bp::_value)>
{ };

bp::terminal<float>::type f = { 3.14 };
std::cout « FloatTerminal()(f);
“3.14 @ 0x7fff0d839aa0”
```

Activating a grammar

```
struct ToString : bp::callable
{
  typedef std::string result_type;

  template <typename T>
  result_type operator()(const T& t)
  {
    return str(boost::format("%s @ %p") % t % &t);
  }
};

struct FloatTerminal
  : bp::when<bp::terminal<float>, ToString(bp::_value)>
{ };

bp::terminal<float>::type f = { 3.14 };
std::cout « FloatTerminal()(f);
“3.14 @ 0x7fff0d839aa0”
```

Activating a grammar

```
struct ToString : bp::callable
{
  typedef std::string result_type;

  template <typename T>
  result_type operator()(const T& t)
  {
    return str(boost::format("%s @ %p") % t % &t);
  }
};

struct FloatTerminal
  : bp::when<bp::terminal<float>, ToString(bp::_value)>
{ };

bp::terminal<float>::type f = { 3.14 };
std::cout « FloatTerminal()(f);
“3.14 @ 0x7fff0d839aa0”
```

Activating a grammar

```
struct ToString : bp::callable
{
  typedef std::string result_type;

  template <typename T>
  result_type operator()(const T& t)
  {
    return str(boost::format("%s @ %p") % t % &t);
  }
};

struct FloatTerminal
  : bp::when<bp::terminal<float>, ToString(bp::_value)>
{ };

bp::terminal<float>::type f = { 3.14 };
std::cout « FloatTerminal()(f);
“3.14 @ 0x7fff0d839aa0”
```

Activating a grammar

```
struct ToString : bp::callable
{
  typedef std::string result_type;

  template <typename T>
  result_type operator()(const T& t)
  {
    return str(boost::format("%s @ %p") % t % &t);
  }
};

struct FloatTerminal
  : bp::when<bp::terminal<float>, ToString(bp::_value)>
{ };

bp::terminal<float>::type f = { 3.14 };
std::cout « FloatTerminal()(f);
“3.14 @ 0x7fff0d839aa0”
```

Activating a grammar

```
struct ToString : bp::callable
{
  typedef std::string result_type;

  template <typename T>
  result_type operator()(const T& t)
  {
    return str(boost::format("%s @ %p") % t % &t);
  }
};

struct FloatTerminal
  : bp::when<bp::terminal<float>, ToString(bp::_value)>
{ };

bp::terminal<float>::type f = { 3.14 };
std::cout « FloatTerminal()(f);
“3.14 @ 0x7fff0d839aa0”
```

Activating a grammar

```
struct ToString : bp::callable
{
  typedef std::string result_type;

  template <typename T>
  result_type operator()(const T& t)
  {
    return str(boost::format("%s @ %p") % t % &t);
  }
};

struct FloatTerminal
  : bp::when<bp::terminal<float>, ToString(bp::_value)>
{ };

bp::terminal<float>::type f = { 3.14 };
std::cout « FloatTerminal()(f);
“3.14 @ 0x7fff0d839aa0”
```

Activating a grammar

```
struct ToString : bp::callable
{
  typedef std::string result_type;

  template <typename T>
  result_type operator()(const T& t)
  {
    return str(boost::format("%s @ %p") % t % &t);
  }
};

struct FloatTerminal
  : bp::when<bp::terminal<float>, ToString(bp::_value)>
{ };

bp::terminal<float>::type f = { 3.14 };
std::cout « FloatTerminal()(f);
“3.14 @ 0x7fff0d839aa0”
```

Activating a grammar

```
struct ToString : bp::callable
{
  typedef std::string result_type;

  template <typename T>
  result_type operator()(const T& t)
  {
    return str(boost::format("%s @ %p") % t % &t);
  }
};

struct FloatTerminal
  : bp::when<bp::terminal<float>, ToString(bp::_value)>
{ };

bp::terminal<float>::type f = { 3.14 };
std::cout « FloatTerminal()(f);
“3.14 @ 0x7fff0d839aa0”
```

Activating a grammar

```
struct ToString : bp::callable
{
  typedef std::string result_type;

  template <typename T>
  result_type operator()(const T& t)
  {
    return str(boost::format("%s @ %p") % t % &t);
  }
};

struct FloatTerminal
  : bp::when<bp::terminal<float>, ToString(bp::_value)>
{ };

bp::terminal<float>::type f = { 3.14 };
std::cout « FloatTerminal()(f);
“3.14 @ 0x7fff0d839aa0”
```

Activating a grammar

```
struct ToString : bp::callable
{
  typedef std::string result_type;

  template <typename T>
  result_type operator()(const T& t)
  {
    return str(boost::format("%s @ %p") % t % &t);
  }
};

struct FloatTerminal
  : bp::when<bp::terminal<float>, ToString(bp::_value)>
{ };

bp::terminal<float>::type f = { 3.14 };
std::cout « FloatTerminal()(f);
“3.14 @ 0x7fff0d839aa0”
```

Activating a grammar

```
struct ToString : bp::callable
{
  typedef std::string result_type;

  template <typename T>
  result_type operator()(const T& t)
  {
    return str(boost::format("%s @ %p") % t % &t);
  }
};

struct FloatTerminal
  : bp::when<bp::terminal<float>, ToString(bp::_value)>
{ };

bp::terminal<float>::type f = { 3.14 };
std::cout « FloatTerminal()(f);
“3.14 @ 0x7fff0d839aa0”
```

proto::when

```
// a + 777.0f

struct ArrayTerminal
  : bp::when<bp::terminal<array_impl>, ToString(bp::_value)>
{ };

struct FloatTerminal
  : bp::when<bp::terminal<float>, ToString(bp::_value)>
{ };

struct Grammar :
  bp::or_<ArrayTerminal,
          FloatTerminal,
          bp::when<bp::plus<bp::terminal<array_impl>,
                            bp::terminal<float> >,
                   ToString(bp::tag::plus(),
                            ToString(bp::_value(bp::_left)),
                            ToString(bp::_value(bp::_right)))>
    >
{ };
```

proto::when

```
// a + 777.0f

struct ArrayTerminal
  : bp::when<bp::terminal<array_impl>, ToString(bp::_value)>
{ };

struct FloatTerminal
  : bp::when<bp::terminal<float>, ToString(bp::_value)>
{ };

struct Grammar :
  bp::or_<ArrayTerminal,
          FloatTerminal,
          bp::when<bp::plus<bp::terminal<array_impl>,
                            bp::terminal<float> >,
                   ToString(bp::tag::plus(),
                            ToString(bp::_value(bp::_left)),
                            ToString(bp::_value(bp::_right)))>
    >
{ };
```

proto::when

```
// a + 777.0f

struct ArrayTerminal
  : bp::when<bp::terminal<array_impl>, ToString(bp::_value)>
{ };

struct FloatTerminal
  : bp::when<bp::terminal<float>, ToString(bp::_value)>
{ };

struct Grammar :
  bp::or_<ArrayTerminal,
          FloatTerminal,
          bp::when<bp::plus<bp::terminal<array_impl>,
                            bp::terminal<float> >,
                   ToString(bp::tag::plus(),
                            ToString(bp::_value(bp::_left)),
                            ToString(bp::_value(bp::_right)))>
    >
{ };
```

proto::when

```
// a + 777.0f

struct ArrayTerminal
  : bp::when<bp::terminal<array_impl>, ToString(bp::_value)>
{ };

struct FloatTerminal
  : bp::when<bp::terminal<float>, ToString(bp::_value)>
{ };

struct Grammar :
  bp::or_<ArrayTerminal,
          FloatTerminal,
          bp::when<bp::plus<bp::terminal<array_impl>,
                            bp::terminal<float> >,
                   ToString(bp::tag::plus(),
                            ToString(bp::_value(bp::_left)),
                            ToString(bp::_value(bp::_right)))>
    >
{ };
```

proto::when

```
// a + 777.0f

struct ArrayTerminal
  : bp::when<bp::terminal<array_impl>, ToString(bp::_value)>
{ };

struct FloatTerminal
  : bp::when<bp::terminal<float>, ToString(bp::_value)>
{ };

struct Grammar :
  bp::or_<ArrayTerminal,
          FloatTerminal,
          bp::when<bp::plus<bp::terminal<array_impl>,
                            bp::terminal<float> >,
                   ToString(bp::tag::plus(),
                            ToString(bp::_value(bp::_left)),
                            ToString(bp::_value(bp::_right)))>
    >
{ };
```

proto::when

```
// a + 777.0f

struct ArrayTerminal
  : bp::when<bp::terminal<array_impl>, ToString(bp::_value)>
{ };

struct FloatTerminal
  : bp::when<bp::terminal<float>, ToString(bp::_value)>
{ };

struct Grammar :
  bp::or_<ArrayTerminal,
          FloatTerminal,
          bp::when<bp::plus<ArrayTerminal,
                            bp::terminal<float> >,
                   ToString(bp::tag::plus(),
                            ToString(bp::_value(bp::_left)),
                            ToString(bp::_value(bp::_right)))>
    >
{ };
```

proto::when

```
// a + 777.0f

struct ArrayTerminal
  : bp::when<bp::terminal<array_impl>, ToString(bp::_value)>
{ };

struct FloatTerminal
  : bp::when<bp::terminal<float>, ToString(bp::_value)>
{ };

struct Grammar :
  bp::or_<ArrayTerminal,
          FloatTerminal,
          bp::when<bp::plus<ArrayTerminal,
                            bp::terminal<float> >,
                   ToString(bp::tag::plus(),
                            ToString(bp::_value(bp::_left)),
                            ToString(bp::_value(bp::_right)))>
    >
{ };
```

proto::when

```
// a + 777.0f

struct ArrayTerminal
  : bp::when<bp::terminal<array_impl>, ToString(bp::_value)>
{ };

struct FloatTerminal
  : bp::when<bp::terminal<float>, ToString(bp::_value)>
{ };

struct Grammar :
  bp::or_<ArrayTerminal,
          FloatTerminal,
          bp::when<bp::plus<ArrayTerminal,
                            FloatTerminal>,
                   ToString(bp::tag::plus(),
                            ToString(bp::_value(bp::_left)),
                            ToString(bp::_value(bp::_right)))>
    >
{ };
```

proto::when

```
// a + 777.0f

struct ArrayTerminal
  : bp::when<bp::terminal<array_impl>, ToString(bp::_value)>
{ };

struct FloatTerminal
  : bp::when<bp::terminal<float>, ToString(bp::_value)>
{ };

struct Grammar :
  bp::or_<ArrayTerminal,
          FloatTerminal,
          bp::when<bp::plus<ArrayTerminal,
                            FloatTerminal>,
                   ToString(bp::tag::plus(),
                            ToString(bp::_value(bp::_left)),
                            ToString(bp::_value(bp::_right)))>
    >
{ };
```

proto::when

```
// a + 777.0f

struct ArrayTerminal
  : bp::when<bp::terminal<array_impl>, ToString(bp::_value)>
{ };

struct FloatTerminal
  : bp::when<bp::terminal<float>, ToString(bp::_value)>
{ };

struct Grammar :
  bp::or_<ArrayTerminal,
          FloatTerminal,
          bp::when<bp::plus<ArrayTerminal,
                            FloatTerminal>,
                   ToString(bp::tag::plus(),
                            ToString(bp::_value(bp::_left)),
                            ToString(bp::_value(bp::_right)))>
    >
{ };
```

proto::when

```
// a + 777.0f

struct ArrayTerminal
  : bp::when<bp::terminal<array_impl>, ToString(bp::_value)>
{ };

struct FloatTerminal
  : bp::when<bp::terminal<float>, ToString(bp::_value)>
{ };

struct Grammar :
  bp::or_<ArrayTerminal,
          FloatTerminal,
          bp::when<bp::plus<ArrayTerminal,
                            FloatTerminal>,
                   ToString(bp::tag::plus(),
                            ToString(bp::_value(bp::_left)),
                            ToString(bp::_value(bp::_right)))>
    >
{ };      // struct plus { };
```

proto::when

```
// a + 777.0f

struct ArrayTerminal
  : bp::when<bp::terminal<array_impl>, ToString(bp::_value)>
{ };

struct FloatTerminal
  : bp::when<bp::terminal<float>, ToString(bp::_value)>
{ };

struct Grammar :
  bp::or_<ArrayTerminal,
          FloatTerminal,
          bp::when<bp::plus<ArrayTerminal,
                            FloatTerminal>,
                   ToString(bp::tag::plus(),
                            ToString(bp::_value(bp::_left)),
                            ToString(bp::_value(bp::_right)))>
    >
{ };
```

proto::when

```
// a + 777.0f

struct ArrayTerminal
  : bp::when<bp::terminal<array_impl>, ToString(bp::_value)>
{ };

struct FloatTerminal
  : bp::when<bp::terminal<float>, ToString(bp::_value)>
{ };

struct Grammar :
  bp::or_<ArrayTerminal,
          FloatTerminal,
          bp::when<bp::plus<ArrayTerminal,
                            FloatTerminal>,
                   ToString(bp::tag::plus(),
                            ToString(bp::_value(bp::_left)),
                            ToString(bp::_value(bp::_right)))>
    >
{ };
```

proto::when

```
// a + 777.0f

struct ArrayTerminal
  : bp::when<bp::terminal<array_impl>, ToString(bp::_value)>
{ };

struct FloatTerminal
  : bp::when<bp::terminal<float>, ToString(bp::_value)>
{ };

struct Grammar :
  bp::or_<ArrayTerminal,
          FloatTerminal,
          bp::when<bp::plus<ArrayTerminal,
                            FloatTerminal>,
                   ToString(bp::tag::plus(),
                            ToString(bp::_value(bp::_left)),
                            ToString(bp::_value(bp::_right)))>
    >
{ };
```

proto::when

```
// a + 777.0f

struct ArrayTerminal
  : bp::when<bp::terminal<array_impl>, ToString(bp::_value)>
{ };

struct FloatTerminal
  : bp::when<bp::terminal<float>, ToString(bp::_value)>
{ };

struct Grammar :
  bp::or_<ArrayTerminal,
          FloatTerminal,
          bp::when<bp::plus<ArrayTerminal,
                            FloatTerminal>,
                   ToString(bp::tag::plus(),
                            ToString(bp::_value(bp::_left)),
                            ToString(bp::_value(bp::_right)))>
    >
{ };
```

proto::when

```
// a + 777.0f

struct ArrayTerminal
  : bp::when<bp::terminal<array_impl>, ToString(bp::_value)>
{ };

struct FloatTerminal
  : bp::when<bp::terminal<float>, ToString(bp::_value)>
{ };

struct Grammar :
  bp::or_<ArrayTerminal,
          FloatTerminal,
          bp::when<bp::plus<ArrayTerminal,
                            FloatTerminal>,
                   ToString(bp::tag::plus(),
                            ArrayTerminal(bp::_left),
                            ToString(bp::_value(bp::_right)))>
    >
{ };
```

proto::when

```
// a + 777.0f

struct ArrayTerminal
  : bp::when<bp::terminal<array_impl>, ToString(bp::_value)>
{ };

struct FloatTerminal
  : bp::when<bp::terminal<float>, ToString(bp::_value)>
{ };

struct Grammar :
  bp::or_<ArrayTerminal,
          FloatTerminal,
          bp::when<bp::plus<ArrayTerminal,
                            FloatTerminal>,
                   ToString(bp::tag::plus(),
                            ArrayTerminal(bp::_left),
                            FloatTerminal(bp::_right))>
    >
{ };
```

They recurse!

```
// (a + 777.0f) + (61.0f + a)

struct ArrayTerminal
  : bp::when<bp::terminal<array_impl>, ToString(bp::_value)>
{ };

struct FloatTerminal
  : bp::when<bp::terminal<float>, ToString(bp::_value)>
{ };

struct Grammar :
  bp::or_<ArrayTerminal,
          FloatTerminal,
          bp::when<bp::plus<ArrayTerminal,
                            FloatTerminal>,
                   ToString(bp::tag::plus(),
                            ArrayTerminal(bp::_left),
                            FloatTerminal(bp::_right))>
     >
{ };
```

They recurse!

```
// (a + 777.0f) + (61.0f + a)

struct ArrayTerminal
  : bp::when<bp::terminal<array_impl>, ToString(bp::_value)>
{ };

struct FloatTerminal
  : bp::when<bp::terminal<float>, ToString(bp::_value)>
{ };

struct Grammar :
  bp::or_<ArrayTerminal,
          FloatTerminal,
          bp::when<bp::plus<ArrayTerminal,
                            FloatTerminal>,
                   ToString(bp::tag::plus(),
                            ArrayTerminal(bp::_left),
                            FloatTerminal(bp::_right))>
     >
{ };
```

They recurse!

```
// (a + 777.0f) + (61.0f + a)

struct ArrayTerminal
  : bp::when<bp::terminal<array_impl>, ToString(bp::_value)>
{ };

struct FloatTerminal
  : bp::when<bp::terminal<float>, ToString(bp::_value)>
{ };

struct Grammar :
  bp::or_<ArrayTerminal,
          FloatTerminal,
          bp::when<bp::plus<ArrayTerminal,
                            FloatTerminal>,
                   ToString(bp::tag::plus(),
                            ArrayTerminal(bp::_left),
                            FloatTerminal(bp::_right))>
     >
{ };
```

They recurse!

```
// (a + 777.0f) + (61.0f + a)

struct ArrayTerminal
  : bp::when<bp::terminal<array_impl>, ToString(bp::_value)>
{ };

struct FloatTerminal
  : bp::when<bp::terminal<float>, ToString(bp::_value)>
{ };

struct Grammar :
  bp::or_<ArrayTerminal,
          FloatTerminal,
          bp::when<bp::plus<Grammar,
                            Grammar>,
                   ToString(bp::tag::plus(),
                            Grammar(bp::_left),
                            Grammar(bp::_right))>
     >
{ };
```

They recurse!

```
// (a + 777.0f) + (61.0f + a)

struct ArrayTerminal
  : bp::when<bp::terminal<array_impl>, ToString(bp::_value)>
{ };

struct FloatTerminal
  : bp::when<bp::terminal<float>, ToString(bp::_value)>
{ };

struct Grammar :
  bp::or_<ArrayTerminal,
          FloatTerminal,
          bp::when<bp::plus<Grammar,
                            Grammar>,
                   ToString(bp::tag::plus(),
                            Grammar(bp::_left),
                            Grammar(bp::_right))>
     >
{ };
```

Tune up the transform

```
struct ToString : bp::callable
{
  typedef std::string result_type;
  // ...
  result_type operator()(bp::tag::plus,
                         const std::string& lhs,
                         const std::string& rhs)
  {
    return str(boost::format("%s PLUS %s") % lhs % rhs );
  }
};

array a;

std::cout « Grammar()(a + 777);
“array_impl @ 0x7fffe0f3f11f PLUS 777 @ 0x7fffe0f3f10c”
```

Tune up the transform

```
struct ToString : bp::callable
{
  typedef std::string result_type;
  // ...
  result_type operator()(bp::tag::plus,
                         const std::string& lhs,
                         const std::string& rhs)
  {
    return str(boost::format("%s PLUS %s") % lhs % rhs );
  }
};

array a;

std::cout « Grammar()(a + 777);
“array_impl @ 0x7fffe0f3f11f PLUS 777 @ 0x7fffe0f3f10c”
```

Tune up the transform

```
struct ToString : bp::callable
{
  typedef std::string result_type;
  // ...
  result_type operator()(bp::tag::plus,
                         const std::string& lhs,
                         const std::string& rhs)
  {
    return str(boost::format("%s PLUS %s") % lhs % rhs );
  }
};

array a;

std::cout « Grammar()(a + 777);
“array_impl @ 0x7fffe0f3f11f PLUS 777 @ 0x7fffe0f3f10c”
```

Tune up the transform

```
struct ToString : bp::callable
{
  typedef std::string result_type;
  // ...
  result_type operator()(bp::tag::plus,
                         const std::string& lhs,
                         const std::string& rhs)
  {
    return str(boost::format("%s PLUS %s") % lhs % rhs );
  }
};

array a;

std::cout « Grammar()(a + 777);
“array_impl @ 0x7fffe0f3f11f PLUS 777 @ 0x7fffe0f3f10c”
```

Kamasu hello world

```
namespace bp = boost::proto;
namespace rk = resophonic::kamasu;

rk::array<float> a(10,10), b(10,10), c;

c = (a / 3.0f) * (b / 7.0f);

struct Array
  : bp::when<bp::terminal<rk::array_impl<float> >,
             bp::_value>
{ };

struct Scalar
  : bp::when<bp::terminal<float>,
             bp::_value>
{ };
```

Kamasu hello world

```
namespace bp = boost::proto;
namespace rk = resophonic::kamasu;

rk::array<float> a(10,10), b(10,10), c;

c = (a / 3.0f) * (b / 7.0f);

struct Array
  : bp::when<bp::terminal<rk::array_impl<float> >,
             bp::_value>
{ };

struct Scalar
  : bp::when<bp::terminal<float>,
             bp::_value>
{ };
```

Kamasu hello world

```
namespace bp = boost::proto;
namespace rk = resophonic::kamasu;

rk::array<float> a(10,10), b(10,10), c;

c = (a / 3.0f) * (b / 7.0f);

struct Array
  : bp::when<bp::terminal<rk::array_impl<float> >,
             bp::_value>
{ };

struct Scalar
  : bp::when<bp::terminal<float>,
             bp::_value>
{ };
```

Kamasu hello world

```
namespace bp = boost::proto;
namespace rk = resophonic::kamasu;

rk::array<float> a(10,10), b(10,10), c;

c = (a / 3.0f) * (b / 7.0f);

struct Array
  : bp::when<bp::terminal<rk::array_impl<float> >,
             bp::_value>
{ };

struct Scalar
  : bp::when<bp::terminal<float>,
             bp::_value>
{ };
```

Kamasu hello world

```
namespace bp = boost::proto;
namespace rk = resophonic::kamasu;

rk::array<float> a(10,10), b(10,10), c;

c = (a / 3.0f) * (b / 7.0f);

struct Grammar
  : bp::or_<bp::when<bp::divides<Array, Scalar>,
                     ArrayScalarOp(bp::tag::divides(),
                                   Array(bp::_left),
                                   Scalar(bp::_right))>,
            bp::when<bp::multiplies<Array, Array>,
                     ArrayArrayOp(bp::tag::multiplies(),
                                  Array(bp::_left),
                                  Array(bp::_right))>
            >
{ };
```

Kamasu hello world

```
namespace bp = boost::proto;
namespace rk = resophonic::kamasu;

rk::array<float> a(10,10), b(10,10), c;

c = (a / 3.0f) * (b / 7.0f);

struct Grammar
  : bp::or_<bp::when<bp::divides<Array, Scalar>,
                     ArrayScalarOp(bp::tag::divides(),
                                   Array(bp::_left),
                                   Scalar(bp::_right))>,
            bp::when<bp::multiplies<Array, Array>,
                     ArrayArrayOp(bp::tag::multiplies(),
                                  Array(bp::_left),
                                  Array(bp::_right))>
            >
{ };
```

Kamasu hello world

```
namespace bp = boost::proto;
namespace rk = resophonic::kamasu;

rk::array<float> a(10,10), b(10,10), c;

c = (a / 3.0f) * (b / 7.0f);

struct Grammar
  : bp::or_<bp::when<bp::divides<Array, Scalar>,
                     ArrayScalarOp(bp::tag::divides(),
                                   Array(bp::_left),
                                   Scalar(bp::_right))>,
            bp::when<bp::multiplies<Array, Array>,
                     ArrayArrayOp(bp::tag::multiplies(),
                                  Array(bp::_left),
                                  Array(bp::_right))>
            >
{ };
```

a / 7.0f: Array-Scalar Transform

```
struct ArrayScalarOp : bp::callable
{
  typedef rk::array_impl<float> result_type;

  template <typename Tag>
  result_type
  operator()(Tag, const rk::array_impl<float>& v,
                  const float& f)
  {
    // hop across the compiler firewall
    transform<float, Tag>(v.data(), v.view_p(), scalar);
    return v;
  }
};
```

.

a / 7.0f: Array-Scalar Transform

```
struct ArrayScalarOp : bp::callable
{
  typedef rk::array_impl<float> result_type;

  template <typename Tag>
  result_type
  operator()(Tag, const rk::array_impl<float>& v,
                   const float& f)
  {
    // hop across the compiler firewall
    transform<float, Tag>(v.data(), v.view_p(), scalar);
    return v;
  }
};
```

.

a / 7.0f: Array-Scalar Transform

```
struct ArrayScalarOp : bp::callable
{
  typedef rk::array_impl<float> result_type;

  template <typename Tag>
  result_type
  operator()(Tag, const rk::array_impl<float>& v,
                   const float& f)
  {
    // hop across the compiler firewall
    transform<float, Tag>(v.data(), v.view_p(), scalar);
    return v;
  }
};
```

.

a / 7.0f: Array-Scalar Transform

```
struct ArrayScalarOp : bp::callable
{
  typedef rk::array_impl<float> result_type;

  template <typename Tag>    // ie. bp::tag::divides
  result_type
  operator()(Tag, const rk::array_impl<float>& v,
                   const float& f)
  {
    // hop across the compiler firewall
    transform<float, Tag>(v.data(), v.view_p(), scalar);
    return v;
  }
};
```

.

a / 7.0f: Array-Scalar Transform

```
struct ArrayScalarOp : bp::callable
{
  typedef rk::array_impl<float> result_type;

  template <typename Tag>
  result_type
  operator()(Tag, const rk::array_impl<float>& v,
                   const float& f)
  {
    // hop across the compiler firewall
    transform<float, Tag>(v.data(), v.view_p(), scalar);
    return v;
  }
};
```

.

a / 7.0f: Array-Scalar Transform

```
struct ArrayScalarOp : bp::callable
{
  typedef rk::array_impl<float> result_type;

  template <typename Tag>
  result_type
  operator()(Tag, const rk::array_impl<float>& v,
                   const float& f)
  {
    // hop across the compiler firewall
    transform<float, Tag>(v.data(), v.view_p(), scalar);
    return v;
  }
};
```

.

a / 7.0f: Array-Scalar Transform

```
struct ArrayScalarOp : bp::callable
{
  typedef rk::array_impl<float> result_type;

  template <typename Tag>
  result_type
  operator()(Tag, const rk::array_impl<float>& v,
                  const float& f)
  {
    // hop across the compiler firewall
    transform<float, Tag>(v.data(), v.view_p(), scalar);
    return v;
  }
};
```

a / 7.0f: Array-Scalar Transform

```
struct ArrayScalarOp : bp::callable
{
  typedef rk::array_impl<float> result_type;

  template <typename Tag>
  result_type
  operator()(Tag, const rk::array_impl<float>& v,
                   const float& f)
  {
    // hop across the compiler firewall
    transform<float, Tag>(v.data(), v.view_p(), scalar);
    return v;
  }
};
```

.

a / 7.0f: Array-Scalar Transform

```
struct ArrayScalarOp : bp::callable
{
  typedef rk::array_impl<float> result_type;

  template <typename Tag>
  result_type
  operator()(Tag, const rk::array_impl<float>& v,
                  const float& f)
  {
    // hop across the compiler firewall
    transform<float, Tag>(v.data(), v.view_p(), scalar);
    return v;
  }
};
```

.

a / 7.0f: calculate gridsize, call kernel

```
template <typename T, typename Tag>
void
transform(T* data, const view_params& vp, T scalar)
{
   bd_t bd = gridsize(vp.linear_size, 64);
   transform_knl<T, Tag><<<bd.first,
                           bd.second>>>(data + vp.offset,
                                        vp, scalar);
}
```

a / 7.0f: calculate gridsize, call kernel

```
template <typename T, typename Tag>
void
transform(T* data, const view_params& vp, T scalar)
{
   bd_t bd = gridsize(vp.linear_size, 64);
   transform_knl<T, Tag><<<bd.first,
                           bd.second>>>(data + vp.offset,
                                        vp, scalar);
}
```

a / 7.0f: calculate gridsize, call kernel

```
template <typename T, typename Tag>
void
transform(T* data, const view_params& vp, T scalar)
{
   bd_t bd = gridsize(vp.linear_size, 64);
   transform_knl<T, Tag><<<bd.first,
                           bd.second>>>(data + vp.offset,
                                        vp, scalar);
}
```

a / 7.0f: calculate gridsize, call kernel

```
template <typename T, typename Tag>
void
transform(T* data, const view_params& vp, T scalar)
{
   bd_t bd = gridsize(vp.linear_size, 64);
   transform_knl<T, Tag><<<bd.first,
                           bd.second>>>(data + vp.offset,
                                        vp, scalar);
}
```

a / 7.0f: calculate gridsize, call kernel

```
template <typename T, typename Tag>
void
transform(T* data, const view_params& vp, T scalar)
{
   bd_t bd = gridsize(vp.linear_size, 64);
   transform_knl<T, Tag><<<bd.first,
                           bd.second>>>(data + vp.offset,
                                        vp, scalar);
}
```

out on the device

```
template <typename T, typename Tag>
__global__ void
transform_knl(T* data, view_params vp, T scalar)
{
  unsigned li = linear_index(threadIdx, /* ... etc */);
  if (li >= vp.linear_size) return;

  unsigned off = actual_index(li, vp.nd, vp.factors, vp.stride

  op_impl_<T, Tag>::impl(data + off, scalar);
}

template <typename T>
struct op_impl_<T, boost::proto::tag::plus>
{
  static void impl(T* t, const T& scalar)
  {
    *t += scalar;
  }
};
```

out on the device

```
template <typename T, typename Tag>
__global__ void
transform_knl(T* data, view_params vp, T scalar)
{
  unsigned li = linear_index(threadIdx, /* ... etc */);
  if (li >= vp.linear_size) return;

  unsigned off = actual_index(li, vp.nd, vp.factors, vp.stride

  op_impl_<T, Tag>::impl(data + off, scalar);
}

template <typename T>
struct op_impl_<T, boost::proto::tag::plus>
{
  static void impl(T* t, const T& scalar)
  {
    *t += scalar;
  }
};
```

out on the device

```
template <typename T, typename Tag>
__global__ void
transform_knl(T* data, view_params vp, T scalar)
{
  unsigned li = linear_index(threadIdx, /* ... etc */);
  if (li >= vp.linear_size) return;

  unsigned off = actual_index(li, vp.nd, vp.factors, vp.stride

  op_impl_<T, Tag>::impl(data + off, scalar);
}

template <typename T>
struct op_impl_<T, boost::proto::tag::plus>
{
  static void impl(T* t, const T& scalar)
  {
    *t += scalar;
  }
};
```

out on the device

```
template <typename T, typename Tag>
__global__ void
transform_knl(T* data, view_params vp, T scalar)
{
  unsigned li = linear_index(threadIdx, /* ... etc */);
  if (li >= vp.linear_size) return;

  unsigned off = actual_index(li, vp.nd, vp.factors, vp.stride

  op_impl_<T, Tag>::impl(data + off, scalar);
}

template <typename T>
struct op_impl_<T, boost::proto::tag::plus>
{
  static void impl(T* t, const T& scalar)
  {
    *t += scalar;
  }
};
```

out on the device

```
template <typename T, typename Tag>
__global__ void
transform_knl(T* data, view_params vp, T scalar)
{
  unsigned li = linear_index(threadIdx, /* ... etc */);
  if (li >= vp.linear_size) return;

  unsigned off = actual_index(li, vp.nd, vp.factors, vp.stride

  op_impl_<T, Tag>::impl(data + off, scalar);
}

template <typename T>
struct op_impl_<T, boost::proto::tag::plus>
{
  static void impl(T* t, const T& scalar)
  {
    *t += scalar;
  }
};
```

Historical curiosity

```
__global__ void
kamasu_elementwise_array_scalar_/*OP*/_/*N*/_knl
(float* data,
 unsigned linear_size,
 /*’, ’.join([’const std::size_t factor%d’ % x for x in range(
 /*’, ’.join([’const int stride%d’ % x for x in range(N)])*/,
 float scalar)
{
  if (INDEX >= linear_size)
    return;

  unsigned actual_index =
    /* ’ + ’.join([’INDEX/factor%d*stride%d’ % (N-1, N-1)]
                  + [’ unsigned(INDEX %% factor%d)/factor%d*st
                     % (n+1,n,n) for n in range(N-1)]) */;
  ...
}
```

Historical curiosity

```
__global__ void
kamasu_elementwise_array_scalar_/*OP*/_/*N*/_knl
(float* data,
 unsigned linear_size,
 /*', '.join(['const std::size_t factor%d' % x for x in range(
 /*', '.join(['const int stride%d' % x for x in range(N)])*/,
 float scalar)
{
  if (INDEX >= linear_size)
    return;

  unsigned actual_index =
    /* ' + '.join(['INDEX/factor%d*stride%d' % (N-1, N-1)]
                  + [' unsigned(INDEX %% factor%d)/factor%d*st
                     % (n+1,n,n) for n in range(N-1)]) */;
  ...
}
```

Avoiding temporaries 1: emulating rvalues

```
// a = b * 3.0f / c * 4.0f;

struct CopyLValue : bp::callable
{
  typedef array_impl<float> result_type;

  result_type
  operator()(const array_impl<float>& a)
  {
    return a.clone();
  }
};

struct RkArrayTerminal
  : bp::when<bp::terminal<rk::array_impl<float> >,
             CopyLValue(bp::_value)>
{ };

struct Grammar : bp::or_<RkArrayTerminal, Scalar, ...
```

Avoiding temporaries 1: emulating rvalues

```
// a = b * 3.0f / c * 4.0f;

struct CopyLValue : bp::callable
{
  typedef array_impl<float> result_type;

  result_type
  operator()(const array_impl<float>& a)
  {
    return a.clone();
  }
};

struct RkArrayTerminal
  : bp::when<bp::terminal<rk::array_impl<float> >,
             CopyLValue(bp::_value)>
{ };

struct Grammar : bp::or_<RkArrayTerminal, Scalar, ...
```

Avoiding temporaries 1: emulating rvalues

```
// a = b * 3.0f / c * 4.0f;

struct CopyLValue : bp::callable
{
  typedef array_impl<float> result_type;

  result_type
  operator()(const array_impl<float>& a)
  {
    return a.clone();
  }
};

struct RkArrayTerminal
  : bp::when<bp::terminal<rk::array_impl<float> >,
             CopyLValue(bp::_value)>
{ };

struct Grammar : bp::or_<RkArrayTerminal, Scalar, ...
```

Avoiding temporaries 1: emulating rvalues

```
// a = b * 3.0f / c * 4.0f;

struct CopyLValue : bp::callable
{
  typedef array_impl<float> result_type;

  result_type
  operator()(const array_impl<float>& a)
  {
    return a.clone();
  }
};

struct RkArrayTerminal
  : bp::when<bp::terminal<rk::array_impl<float> >,
             CopyLValue(bp::_value)>
{ };

struct Grammar : bp::or_<RkArrayTerminal, Scalar, ...
```

Avoiding temporaries 1: emulating rvalues

```
// a = b * 3.0f / c * 4.0f;

struct CopyLValue : bp::callable
{
  typedef array_impl<float> result_type;

  result_type
  operator()(const array_impl<float>& a)
  {
    return a.clone();
  }
};

struct RkArrayTerminal
  : bp::when<bp::terminal<rk::array_impl<float> >,
             CopyLValue(bp::_value)>
{ };

struct Grammar : bp::or_<RkArrayTerminal, Scalar, ...
```

Avoiding temporaries 1: emulating rvalues

```
// a = b * 3.0f / c * 4.0f;

struct CopyLValue : bp::callable
{
  typedef array_impl<float> result_type;

  result_type
  operator()(const array_impl<float>& a)
  {
    return a.clone();
  }
};

struct RkArrayTerminal
  : bp::when<bp::terminal<rk::array_impl<float> >,
             CopyLValue(bp::_value)>
{ };

struct Grammar : bp::or_<RkArrayTerminal, Scalar, ...
```

Avoiding temporaries 2: reuse the LHS

```
// a = sin(a);

template <typename Expr>
array<T>::operator=(Expr const& expr)
{
  array_impl<T> result = Grammar()(expr);
  self_.copy_from(result);
}
```

Avoiding temporaries 2: reuse the LHS

```
// a = sin(a);

template <typename Expr>
array<T>::operator=(Expr const& expr)
{
  array_impl<T> result = Grammar()(expr);
  self_.copy_from(result);
}
```

Avoiding temporaries 2: reuse the LHS

```
// a = sin(a);

struct data_t { array_impl<float>* tmp; };

template <typename Expr>
array<float>::operator=(Expr const& expr)
{
  data_t data; data.tmp = this->base_ptr();
  array_impl<float> tmp = Grammar()(expr, bool(), data);
  self_.copy_from(result);
}

CopyLValue::result_type
CopyLValue::operator()(const array_impl<float>& a, data_t& dat
{
  if (data.tmp == &a) { data.tmp = 0; return a; }
  else                { return a.clone();       }
}
```

Avoiding temporaries 2: reuse the LHS

```
// a = sin(a);

struct data_t { array_impl<float>* tmp; };

template <typename Expr>
array<float>::operator=(Expr const& expr)
{
  data_t data; data.tmp = this->base_ptr();
  array_impl<float> tmp = Grammar()(expr, bool(), data);
  self_.copy_from(result);
}

CopyLValue::result_type
CopyLValue::operator()(const array_impl<float>& a, data_t& dat
{
  if (data.tmp == &a) { data.tmp = 0; return a; }
  else                { return a.clone();       }
}
```

komrade: norm of vector

```
template <typename T>
struct square {
  __host__ __device__
  T operator()(const T& x) const
  {
    return x * x;
  }
};

float x[4] = { 1.0, 2.0, 3.0, 4.0 };
komrade::device_vector<float> d_x(x, x + 4);

square<float>        unary_op;
komrade::plus<float> binary_op;
float init = 0;

sqrt( komrade::transform_reduce(d_x.begin(), d_x.end(),
                                unary_op, init, binary_op) );
```

komrade: norm of vector

```
template <typename T>
struct square {
  __host__ __device__
  T operator()(const T& x) const
  {
    return x * x;
  }
};

float x[4] = { 1.0, 2.0, 3.0, 4.0 };
komrade::device_vector<float> d_x(x, x + 4);

square<float>        unary_op;
komrade::plus<float> binary_op;
float init = 0;

sqrt( komrade::transform_reduce(d_x.begin(), d_x.end(),
                                unary_op, init, binary_op) );
```

komrade: norm of vector

```
template <typename T>
struct square {
  __host__ __device__
  T operator()(const T& x) const
  {
    return x * x;
  }
};

float x[4] = { 1.0, 2.0, 3.0, 4.0 };
komrade::device_vector<float> d_x(x, x + 4);

square<float>        unary_op;
komrade::plus<float> binary_op;
float init = 0;

sqrt( komrade::transform_reduce(d_x.begin(), d_x.end(),
                                unary_op, init, binary_op) );
```

komrade: norm of vector

```
template <typename T>
struct square {
  __host__ __device__
  T operator()(const T& x) const
  {
    return x * x;
  }
};

float x[4] = { 1.0, 2.0, 3.0, 4.0 };
komrade::device_vector<float> d_x(x, x + 4);

square<float>        unary_op;
komrade::plus<float> binary_op;
float init = 0;

sqrt( komrade::transform_reduce(d_x.begin(), d_x.end(),
                                unary_op, init, binary_op) );
```

komrade: norm of vector

```
template <typename T>
struct square {
  __host__ __device__
  T operator()(const T& x) const
  {
    return x * x;
  }
};

float x[4] = { 1.0, 2.0, 3.0, 4.0 };
komrade::device_vector<float> d_x(x, x + 4);

square<float>        unary_op;
komrade::plus<float> binary_op;
float init = 0;

sqrt( komrade::transform_reduce(d_x.begin(), d_x.end(),
                                unary_op, init, binary_op) );
```

komrade: norm of vector

```
template <typename T>
struct square {
  __host__ __device__
  T operator()(const T& x) const
  {
    return x * x;
  }
};

float x[4] = { 1.0, 2.0, 3.0, 4.0 };
komrade::device_vector<float> d_x(x, x + 4);

square<float>        unary_op;
komrade::plus<float> binary_op;
float init = 0;

sqrt( komrade::transform_reduce(d_x.begin(), d_x.end(),
                                unary_op, init, binary_op) );
```

komrade: norm of vector

```
template <typename T>
struct square {
  __host__ __device__
  T operator()(const T& x) const
  {
    return x * x;
  }
};

float x[4] = { 1.0, 2.0, 3.0, 4.0 };
komrade::device_vector<float> d_x(x, x + 4);

square<float>        unary_op;
komrade::plus<float> binary_op;
float init = 0;

sqrt( komrade::transform_reduce(d_x.begin(), d_x.end(),
                                unary_op, init, binary_op) );
```

komrade: norm of vector

```
template <typename T>
struct square {
  __host__ __device__
  T operator()(const T& x) const
  {
    return x * x;
  }
};

float x[4] = { 1.0, 2.0, 3.0, 4.0 };
komrade::device_vector<float> d_x(x, x + 4);

square<float>        unary_op;
komrade::plus<float> binary_op;
float init = 0;

sqrt( komrade::transform_reduce(d_x.begin(), d_x.end(),
                                unary_op, init, binary_op) );
```

komrade: norm of vector

```
template <typename T>
struct square {
  __host__ __device__
  T operator()(const T& x) const
  {
    return x * x;
  }
};

float x[4] = { 1.0, 2.0, 3.0, 4.0 };
komrade::device_vector<float> d_x(x, x + 4);

square<float>        unary_op;
komrade::plus<float> binary_op;
float init = 0;

sqrt( komrade::transform_reduce(d_x.begin(), d_x.end(),
                                unary_op, init, binary_op) );
```

komrade: norm of vector

```
template <typename T>
struct square {
  __host__ __device__
  T operator()(const T& x) const
  {
    return x * x;
  }
};

float x[4] = { 1.0, 2.0, 3.0, 4.0 };
komrade::device_vector<float> d_x(x, x + 4);

square<float>        unary_op;
komrade::plus<float> binary_op;
float init = 0;

sqrt( komrade::transform_reduce(d_x.begin(), d_x.end(),
                                unary_op, init, binary_op) );
```

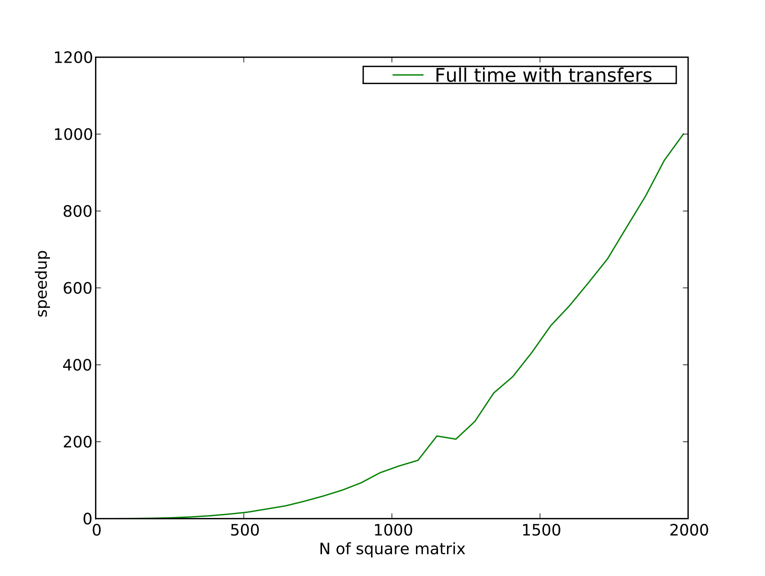
PyCuda (Andreas Klöckner)

```
import pycuda.autoinit, pycuda.driver as drv, numpy

mod = drv.SourceModule("""
__global__ void multiply_them(float *dest, float *a, float *b)
{
  const int i = threadIdx.x;
  dest[i] = a[i] * b[i];
}
""")

multiply_them = mod.get_function("multiply_them")
a = numpy.random.randn(400).astype(numpy.float32)
b = numpy.random.randn(400).astype(numpy.float32)
dest = numpy.zeros_like(a)
multiply_them(
    drv.Out(dest), drv.In(a), drv.In(b),
    block=(400,1,1))
print dest-a*b
```

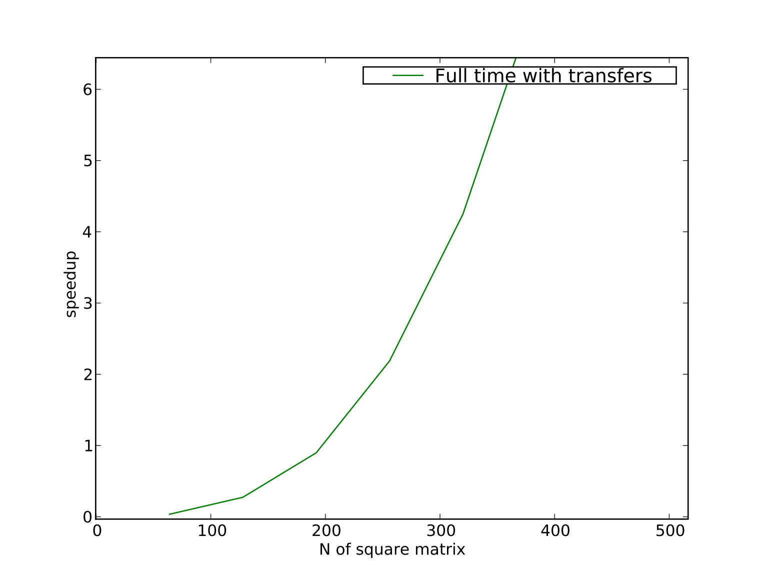
Benchmarks

DIRECTOR OF MARKETECTURE
IT IS BETTER TO SEEM GOOD THAN TO BE GOOD.
BEING GOOD (OVERRATED)
scottadams@aol.com
www.dilbert.com
A MISLEADING BENCHMARK TEST CAN ACCOMPLISH IN MINUTES WHAT YEARS OF GOOD ENGINEERING CAN NEVER DO.
3-2-09 © 2009 Scott Adams, Inc./Dist. by UFS, Inc.
IS IT OUR MATURITY THAT MAKES THAT CONCEPT SOUND OKAY?
I HOPE SO.

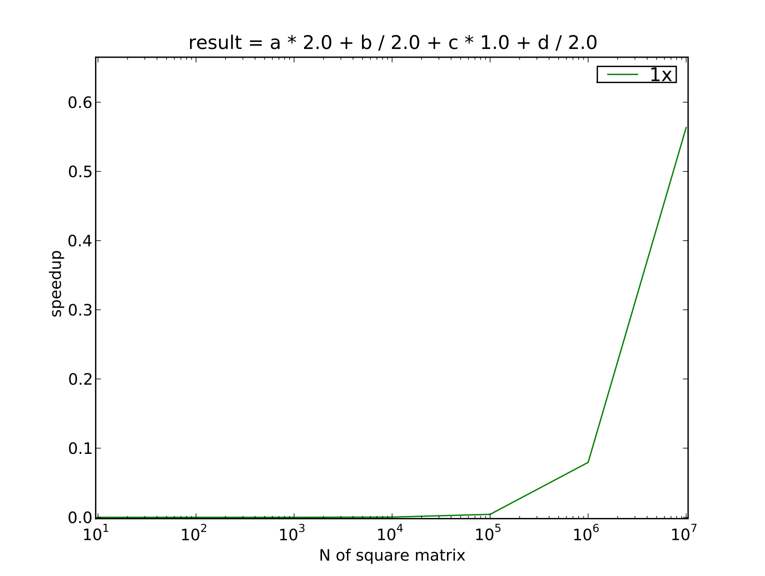
NxN matrices, A = B * C (via cublas)



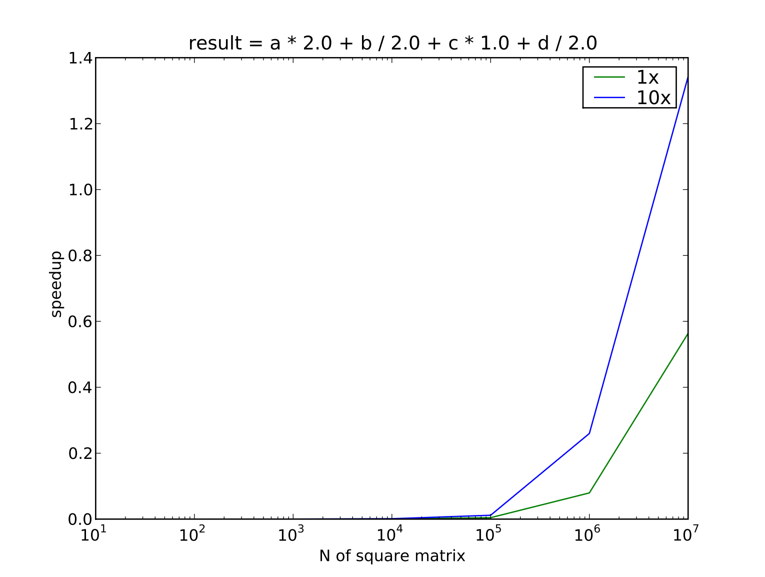
Breakeven @ 200x200 matrices



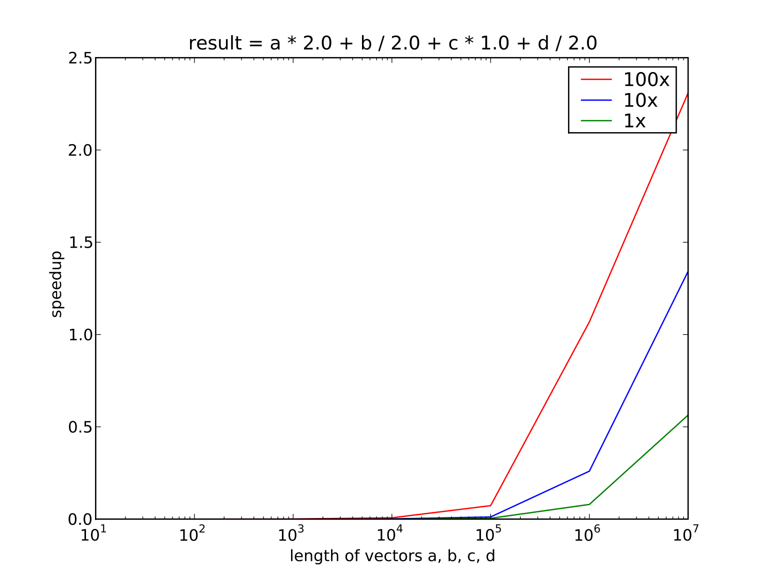
Low numeric intensity == low performance



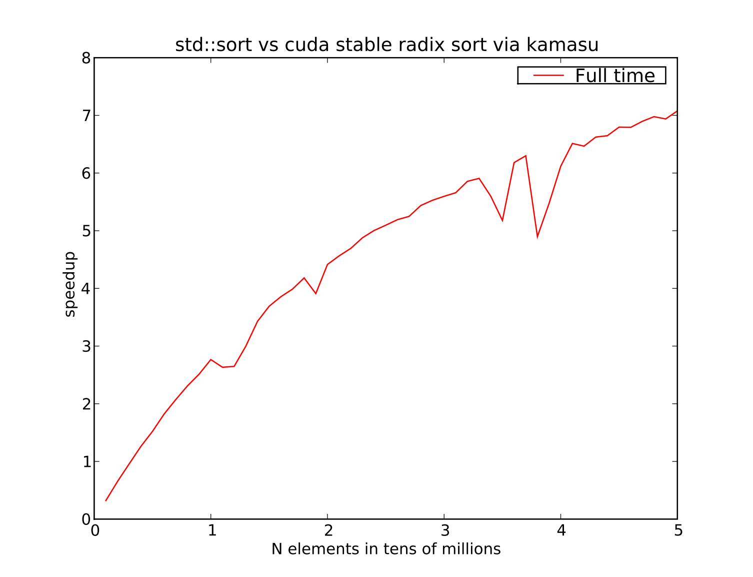
Low numeric intensity == low performance



Low numeric intensity == low performance



Sort – wall clock time including transfers



Summary

- An n-dimensional array type with storage on the device
- Conversions for boost::ublas and std:: types to/from the device
- A grammar for (very) basic linear algebra operations and some primitive functions
- Effort hamstrung to-date by previous version of CUDA compiler, NVIDIA appears to be making good progress
- Nonetheless possible to get a few interesting optimizations via proto
- Even if wallclock time is worse than a simple CPU implementation, you're still freeing up CPU cycles by moving to the GPU
- The big questions are granularity and composition (and "streaming")
- It needs a problem to solve
- Techniques available expected to expand rapidly soon

Questions?